

步进机电有限公司

Step-servo　Co.，　Ltd.

MPC08 运动控制卡

操

作

手

册

（MPC08SP V1.0.3）

目　　录

1 概　　述

1.1 MPC08 的软硬件简介

MPC08 控制卡是基于 PC 机 PCI 总线的步进电机或数字式伺服电机的上位控制单元，它与 PC 机构成主从式控制结构：PC 机负责人机交互界面的管理和控制系统的实时监控等方面的工作（例如键盘和鼠标的管理、系统状态的显示、控制指令的发送、外部信号的监控等等）；MPC08 卡完成运动控制的所有细节（包括脉冲和方向信号的输出、自动升降速的处理、原点和限位等信号的检测等等）。

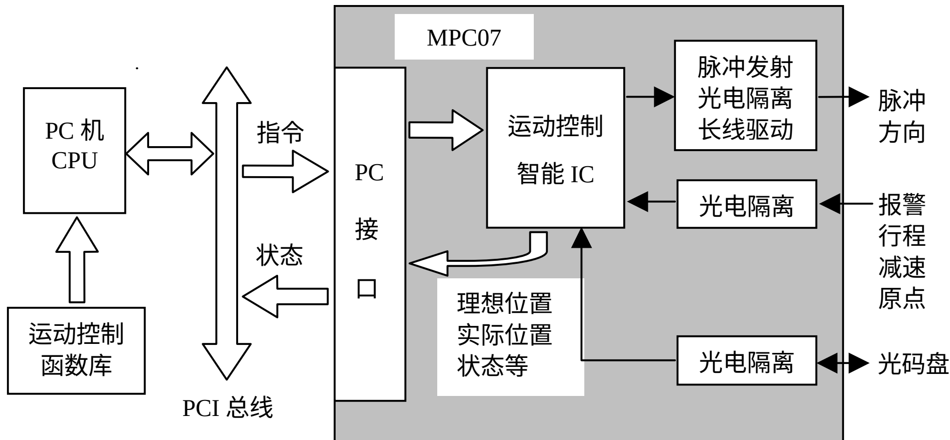
每块 MPC08 卡可控制 4 轴步进电机或数字式伺服电机，并支持多卡共用，以实现多于四个运动轴的控制；每轴均可输出脉冲和方向信号，以控制电机的运转；同时，可外接原点、减速、限位等开关信号，以实现回原点、保护等功能，这些开关信号由 MPC08 卡自动检测并作出反应。

MPC08 卡采用先进的控制芯片，具有梯形升降速曲线，最高输出频率可达 4.0MHz，带有编码器反馈端口，适用于数字式交流伺服系统或闭环的步进电机控制系统。

MPC08 配备了功能强大、内容丰富的 Windows 驱动程序、DLL 函数库及示例程序。MPC08 在插补算法和运动函数的执行效率方面采用了更有效的方法，提高了插补精度、插补速度和实时性。利用 MPC08 的示例程序既可以很快地熟悉 MPC08 控制卡的软、硬件功能，又可以方便快捷地测试执行电机及驱动系统在完成各种运动时的性能特性。MPC08 运动函数库用于二次开发，用户只要用 VC＋＋或 Visual Basic 等支持 Windows 标准 32 位动态链接库（DLL）调用的开发工具编制所需的用户界面程序，并把它与 MPC08 运动库链接起来，就可以开发出自己的控制系统，例如：数控系统、检测设备、自动生产线等。MPC08 的运动函数库能够完成与运动控制有关的复杂细节（比如：升降速、直线插补等），这样就可以大大缩短控制系统的开发周期。

1.2 MPC08 的结构

MPC08 控制卡作为开发运动控制系统的平台，其结构是开放式的。该卡插在 PC 机 PCI 扩展槽内使用，同时使用控制卡的数量和各卡上的控制轴数可方便地配置；MPC08 卡提供了功能强大的运动控制函数库，并可以充分利用 PC 机现有的资源来开发完美的运动控制系统。MPC08 控制卡的结构示意图如下：



MPC08 结构示意图

1.3 MPC08 的技术特性和使用范围

MPC08 控制卡主要特征有：开放式结构、使用简便、功能丰富、可靠性高等。MPC08 的特征体现在硬件和软件两个方面：在硬件方面采用 PC 机的 PCI 总线方式，适用范围广，卡上无需进行任何跳线设置，所有资源自动配置，在 Windows98、Windows2000 及 Windows XP 操作系统中支持即插即用，使用非常方便；MPC08 的接线方式采用 DB62 型插头，可使用屏蔽线缆，并且所有的输入、输出信号均用光电隔离，提高了控制卡的可靠性和抗干扰能力；在软件方面提供了丰富的运动控制函数库，以满足不同的应用要求。用户只需根据控制系统的要求编制人机界面，并调用 MPC08 运动函数库中的指令函数，就可以开发出既满足要求又成本低廉的多轴运动控制系统。

MPC08 的技术指标主要有：

- 4 轴步进或数字式伺服控制；
- 脉冲输出速度可达 4.0Mpps；
- 梯形加减速；
- 运动中可变速；
- 多轴高速线性插补；
- 可输出脉冲/方向或双脉冲信号；
- 可外接编码器位置反馈；
- 每轴带有原点、减速和限位开关等接口；
- 可扩展出 16 路通用输入和 16 路通用输出接口；
- WINDOWS 环境下 WDM、DLL 库；
- 演示程序。

正是由于 MPC08 的开放式结构，使之应用范围十分广泛，在由步进电机和数字式伺服电机组成的基于 PC 机的运动控制系统中，都可以使用 MPC08 作为核心控制单元，例如：

- 数控机床、加工中心、机器人等；
- X-Y-Z 控制台；
- 绘图仪、雕刻机、印刷机械；
- 送料装置、云台；
- 打标机、绕线机；
- 医疗设备；
- 包装机械、纺织机械；

等等。

目前版本主要用于点位控制，无圆弧插补。无批处理方式，系统始终处于立即执行方式。

1.4 MPC08 的运动控制功能

MPC08 控制卡的运动控制功能主要取决于运动函数库。运动函数库为单轴及多轴的步进或伺服控制提供了许多运动函数：单轴运动、多轴独立运动、多轴插补运动等等。另外，为了配合运动控制系统的开发，还提供了间隙补偿功能。下面简单介绍一下这些函数的功能。

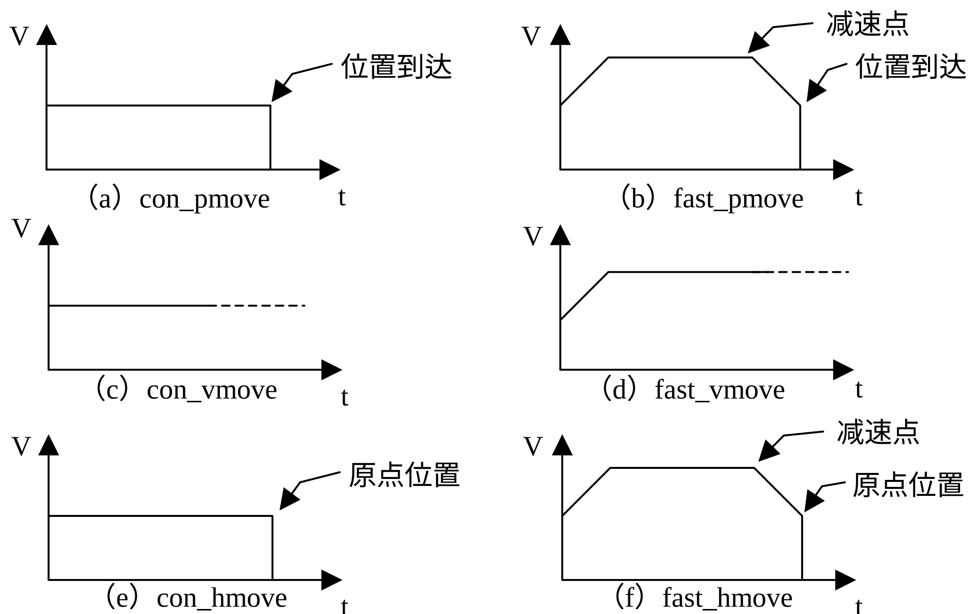
1.4.1 单轴运动控制

单轴运动有三个基本的类型：

- 点位运动（pmove）
- 连续运动（vmove）
- 回原点运动（hmove）

这些运动又可以在常速模式或梯形速度模式下工作，因此，总共有六种基本运动类型，列表如下：

con_pmove	以常速移动指定距离（图（a)）
fast_pmove	以梯形速度移动指定距离（图（b)）
con_vmove	以指定的常速连续运动（图（c)）
fast_vmove	加速后保持在指定高速的连续运动（图（d)）
con_hmove	以常速运动至原点（图（e)）
fast_hmove	加速后快速移至原点位置（图（f)）



运动速度图形

带有升／降速控制的运动函数称之为快速（fast）运动函数，譬如：fast_pmove，fast_vmove 和 fast_hmove，而常速运动函数则称之为常速（con）运动函数，如 con_pmove，con_vmove，con_hmove。

1.4.2 多轴独立运动控制

多个运动轴能以独立的形式进行点位运动、连续运动和回原点运动（同时开始，不一定同时到达）。这类运动一般在函数名的末尾以 2 或 3 来指明参加运动的轴数。例如 con_pmove2 是一个两轴同时独立做点位运动的函数，fast_home3 是三轴独立做回原点运动的函数。这一类函数总共有 24 个。

1.4.3 多轴插补运动控制

多轴插补函数能以特定的矢量速度执行线性插补运动。参与插补运动的各轴同时开始运动，并且按照特定的算法同时到达各自的目标位置。线性插补函数允许两轴或三轴沿直线运动；做直线插补运动时，可以采用均匀矢量速度方式或梯形矢量速度方式。例如，fast_line3 函数让三轴以梯形矢量速度走直线运动。这一类的函数总共有 4 个。

1.4.4 运动指令执行方式

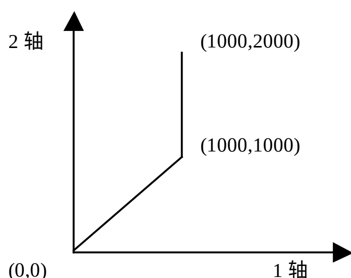
运动指令以立即方式执行：

立即方式主要在单指令实时控制不同的轴运动时使用。该方式下不等上一条运动指令控制的所有轴运动完毕即开始下一条运动指令的执行。若新发出指令控制的轴只要未处于运动状态则立即开始按新运动指令运动，否则新发出指令不予执行并返回一个错误。这种方式若用在多条不同的运动指令连续执行时使用可能造成某些指令无法执行，除非开发人员通过检查运动状态或错误代码加以避免。

以下代码示例说明立即方式指令执行过程：

```
……
con_pmove(1,1000);
con_pmove(2,2000);
con_pmove(1,1000);
……
```

运行后运动轨迹如下：（其中第三条指令执行不到，因为第三条指令发出时第一条指令控制的 1 轴正在运动。）



立即方式

注意：

- 系统始终处于立即执行方式。
- 无批处理方式。
- 无圆弧运动指令。

1.4.5 其它能力

MPC08 的运动函数库也提供还有间隙补偿函数，在机械结构存在间隙时，往复运动的位置精度会受到影响，在电机每次改变方向时应进行间隙补偿。

2 控制卡的安装

2.1 硬件安装

MPC08 控制卡对 PC 机的硬件要求十分简单：能安装 Windows 98、2000、XP 等操作系统，并带有 PCI 插槽的 486 以上机型即可，建议使用更高主频的 Pentium 及以上机型以获取更好的性能。为了整个控制系统的可靠性，建议使用工控 PC 机。

MPC08 卡基于 PCI 总线，因此卡上无需进行跳线设置。

为了保证安全，插卡时应按照下列步骤操作：

1．关 PC 机，并切断电源；
2．打开 PC 机箱，选择未用的 PCI 扩展槽，并插入 MPC08 控制卡；
3．固定 MPC08 控制卡，并盖好 PC 机；
4．连接 MPC08 与电机驱动器等；
5．接上电源，并启动 PC 机。

注意：

在选用普通 PC 机时为避免产生潜在的资源冲突从而导致控制卡驱动程序无法正常加载，建议尽量不要选用集成了声卡、显卡、网卡等多种设备的集成主板。

2.2 软件安装

2.2.1 软件使用要求

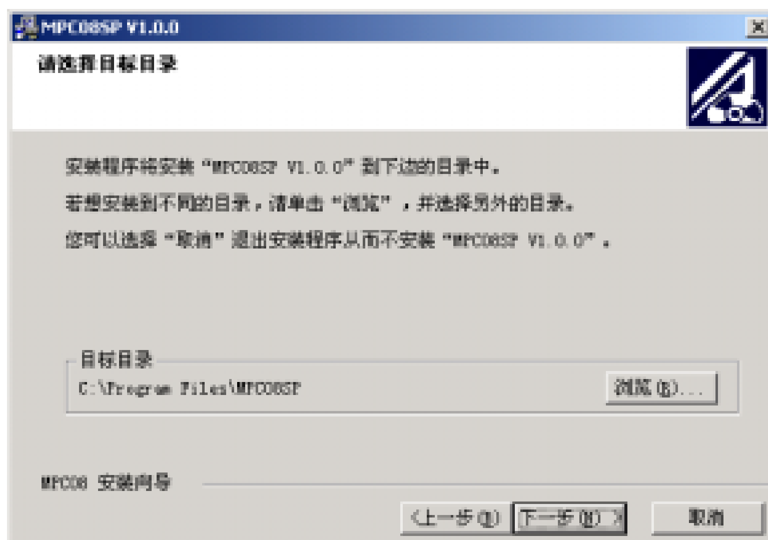
MPC08 控制卡支持 Windows 98、2000、XP 等操作系统。用户可根据自己的软件技术优势进行选择。

MPC08 配有 WINDOWS 环境下的设备驱动程序、运动函数库（以动态链接库的形式提供）和演示软件，以满足不同运动控制系统的开发和测试需要，选择的开发工具只要支持标准的 Windows DLL 调用即可。

2.2.2 软件安装

在 Windows 98、2000、XP 平台下，由于操作系统支持即插即用，当卡正确插入 PCI 插槽，操作系统启动后将会自动检测到 PCI card，此时可按照以下步骤完成驱动程序、函数库以及示例程序的安装。

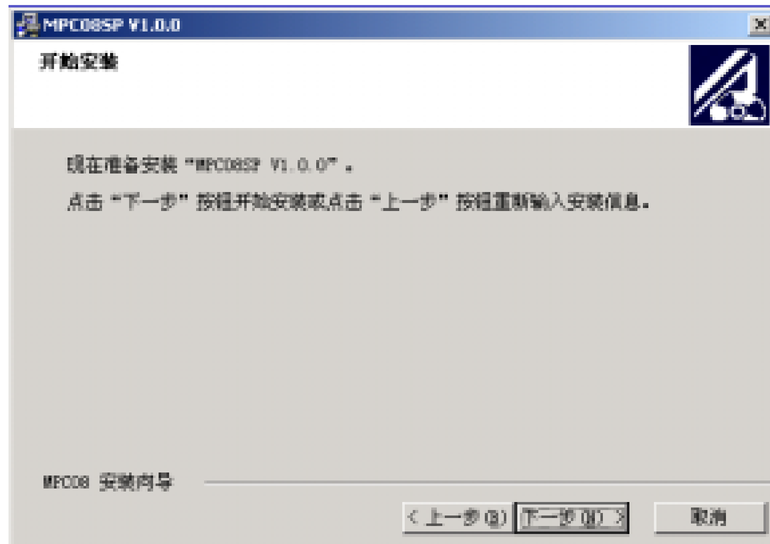
1） 系统检测到 PCI 卡后出现如下画面，此时单击 “ 取消 ”。
2） 运行安装盘根目录下的 “ MPC08SP Setup.exe ”。然后单击 “ 下一步 ”。



3） 单击 " 下一步 "。选择安装模块：驱动程序、应用程序（包含函数库和示例程序），默认情形二者均选中。



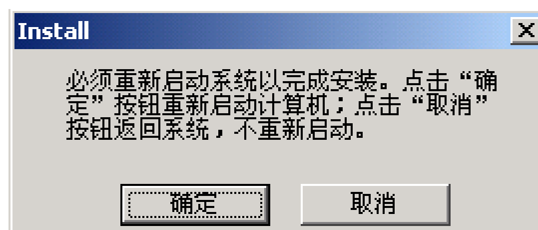
4） 单击 " 下一步 "，开始安装。



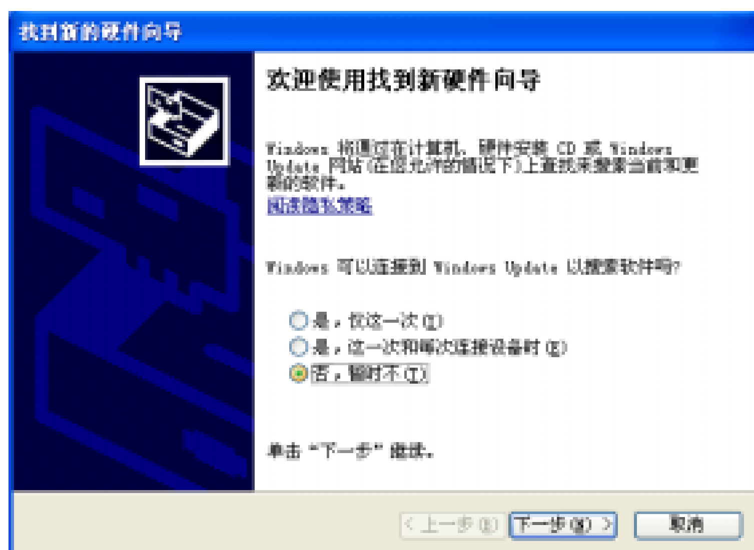
5） 单击“ 完成 ”。



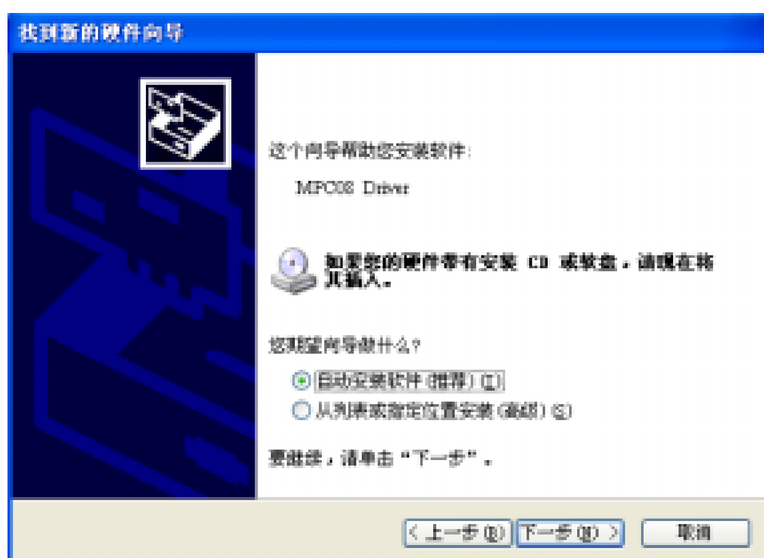
6） 系统提示需要重新启动计算机。选择确定，系统重新启动后即可完成驱动程序的安装。



7） 如果在 Windows XP 下安装 MPC08，重新启动系统后，将出现如下提示，选择第三个选项：“ 否，暂时不 ”，单击下一步。



8） 选择第一个选项：“ 自动安装软件（推荐）（I）”，单击下一步完成安装。



3 MPC08 接口

MPC08 控制卡采用 DB62 接口，外接线可采用屏蔽线缆，以提高控制卡的抗干扰能力。其中开关量信号（原点、减速、限位以及 I/O 信号等）采用 12~24DCV 开关电源；脉冲量信号（脉冲、方向等）采用 5DCV 开关电源。

MPC08 运动控制器由 MPC08 主板和 P62 转接板组成。

3.1 信号接口定义

备注栏："输入"表示该信号为输入信号，"输出"表示该信号为输出信号。

MPC08 卡的 DB62 及 P62 接口定义：

P62 编号	DB62 编号	名称	定义	P62 编号	DB62 编号	名称	定义
D1-1	37	SD1	减速 1	D2-1	16	EL1-	负向限位 1
D1-2	58	EL1+	正向限位 1	D2-2	38	ORG1	原点 1
D1-3	17	SD2	减速 2	D2-3	59	EL2-	负向限位 2
D1-4	39	EL2+	正向限位 2	D2-4	18	ORG2	原点 2
D1-5	60	SD3	减速 3	D2-5	40	EL3-	负向限位 3
D1-6	19	EL3+	正向限位 3	D2-6	61	ORG3	原点 3
D1-7	41	SD4	减速 4	D2-7	20	EL4-	负向限位 4
D1-8	62	EL4+	正向限位 4	D2-8	42	ORG4	原点 4
D1-9	21	ALM	外部报警输入	D2-9	57	EA1-	编码器 A1-
D1-10	15	EA1+	编码器 A1+	D2-10	36	EB1-	编码器 B1-
D1-11	56	EB1+	编码器 B1+	D2-11	14	EZ1-	编码器 Z1-
D1-12	35	EZ1+	编码器 Z1+	D2-12	55	EA2-	编码器 A2-
D1-13	13	EA2+	编码器 A2+	D2-13	34	EB2-	编码器 B2-
D1-14	54	EB2+	编码器 B2+	D2-14	12	EZ2-	编码器 Z2-
D1-15	33	EZ2+	编码器 Z2+	D2-15	53	EA3-	编码器 A3-
D1-16	11	EA3+	编码器 A3+	D2-16	28/48	GND	5/24 V 电源地
D1-17	7	+5V	5V 电源正	D2-17	28/48	GND	5/24 V 电源地
D4-1	49	+24V	24V 电源正	D3-1	28/48	GND	5/24 V 电源地
D4-2	7	+5V	5 V 电源正	D3-2	28/48	GND	5/24 V 电源地
D4-3	52	EB3+	编码器 B3+	D3-3	32	EB3-	编码器 B3-
D4-4	31	EZ3+	编码器 Z3+	D3-4	10	EZ3-	编码器 Z3-
D4-5	9	EA4+	编码器 A4+	D3-5	51	EA4-	编码器 A4-
D4-6	50	EB4+	编码器 B4+	D3-6	30	EB4-	编码器 B4-
D4-7	29	EZ4+	编码器 Z4+	D3-7	8	EZ4-	编码器 Z4-
D4-8	49	+24V	24V 电源正	D3-8	28/48	GND	5/24 V 电源地
D4-9	7	+5V	5 V 电源正	D3-9	28/48	GND	5/24 V 电源地
D4-10	25	DIR1+	方向 1+	D3-10	4	DIR1-	方向 1-
D4-11	46	PUL1+	脉冲 1+	D3-11	26	PUL1-	脉冲 1-
D4-12	5	DIR2+	方向 2+	D3-12	47	DIR2-	方向 2-
D4-13	27	PUL2+	脉冲 2+	D3-13	6	PUL2-	脉冲 2-
D4-14	3	DIR3+	方向 3+	D3-14	45	DIR3-	方向 3-
D4-15	44	PUL3+	脉冲 3+	D3-15	24	PUL3-	脉冲 3-

D4-16	23	DIR4+	方向 4+	D3-16	2	DIR4-	方向 4-
D4-17	1	PUL4+	脉冲 4+	D3-17	43	PUL4-	脉冲 4-

注：信号名称中的 1、2、3、4 分别对应 MPC08 卡的第 1、2、3、4 轴。

各接口信号的详细说明如下：

类型	功能	编号	说　　明
脉冲量信号	脉冲/方向	D3-10~D3-17 D4-10~D4-17	脉冲/方向信号与步进电机驱动器或数字式伺服电机驱动器相连以控制其运转。 脉冲/方向信号为光电隔离的差分式输出信号，以提高其抗干扰能力。对于仅需要单端式信号的驱动器，只要接该差分信号的正端即可（参见接线方法）；对于接收双脉冲信号的驱动器，PUL 端为正转（CW）脉冲输出端，DIR 端为反转（CCW）脉冲输出端（这种情况下，应调用 set_output_mode 设置 MPC08 卡的脉冲输出模式，参见 set_output_mode 函数说明）。
	开关电源	D1-17 D4-2 D4-9	5V 开关电源正。为脉冲/方向提供光电隔离和编码器差分输入提供驱动。该电源由外部提供。外部 5V 电源可接这三个管脚的任何一个。
		D2-16、D2-17 D3-1、D3-2 D3-8、D3-9	5V 开关电源地。与 24V 开关电源共地。外部 5V 电源地可接这几个管脚的任何一个。
开关量信号	限位	D1-2、D1-4 D1-6、D1-8 D2-1、D2-3 D2-5、D2-7	MPC08 卡上每个控制轴有两个限位输入信号（EL+和 EL-）。在 MPC08 卡发送脉冲时，如果接收到相应的限位信号，MPC08 卡将立即停止发送脉冲。
	减速	D1-1、D1-3 D1-5、D1-7	MPC08 卡上每个控制轴有一个减速输入信号（SD）。在 MPC08 卡执行快速指令发送脉冲时，如果接收到相应的减速信号，MPC08 卡将以设定的加速度减速至低速。
	原点	D2-2、D2-4 D2-6、D2-8	MPC08 卡上每个控制轴有一个原点输入信号（ORG）。在 MPC08 卡执行回原点指令发送脉冲时，如果接收到相应的原点信号，即表示已到达原点，MPC08 卡将立即停止发送脉冲。
	外部报警	D1-9	MPC08 卡有一个共用的外部报警输入信号，当 MPC08 卡接收到该信号时，卡上的各轴将立即停止发送脉冲。
	开关电源	D4-1、D4-8	12~24DCV 的开关电源正，该电源由外部提供，为所有开关量信号提供驱动。外部 24V 电源可接这两个管脚的任何一个。
		D2-16、D2-17 D3-1、D3-2 D3-8、D3-9	24V 开关电源地。与 5V 开关电源共地。外部 24V 电源地可接这几个管脚的任何一个。

另外，根据用户需要，可扩展 16 路通用输入和 16 路通用输出接口。这时需要增加通用 I/O 扩展板。利用 40 芯扁平电缆连接 MPC08 主板和通用 I/O 扩展板，通用 I/O 扩展板安装到计算机 PCI 插槽中，但不占用系统资源。通用 I/O 扩展板接口定义如下：

DB37 引脚号	定义	备注	DB37 引脚号	定义	备注
—	—	—	19	通用输入 9	输入
37	24V 电源正	输入	18	通用输入 10	输入
36	24V 电源地	输入	17	通用输入 11	输入
35	5V 电源正	输入	16	通用输入 12	输入
34	5V 电源地	输入	15	通用输入 13	输入

33	通用输入 16	输入	14	通用输入 14	输入
32	通用输入 1	输入	13	通用输入 15	输入
31	通用输入 2	输入	12	通用输出 16	输出
30	通用输入 3	输入	11	通用输出 15	输出
29	通用输入 4	输入	10	通用输出 14	输出
28	通用输入 5	输入	9	通用输出 13	输出
27	通用输入 6	输入	8	通用输出 12	输出
26	通用输入 7	输入	7	通用输出 11	输出
25	通用输入 8	输入	6	通用输出 10	输出
24			5	通用输出 9	输出
23	通用输出 8	输出	4	通用输出 7	输出
22	通用输出 6	输出	3	通用输出 5	输出
21	通用输出 4	输出	2	通用输出 3	输出
20	通用输出 2	输出	1	通用输出 1	输出

与通用 I/O 扩展板相连的 P37 转接板定义如下：

引脚号	定义	备注	引脚号	定义	备注
1	24V 电源地	输入	2	24V 电源正	输入
3	5V 电源地	输入	4	5V 电源正	输入
5	通用输入 1	输入	6	通用输入 5	输入
7	通用输入 16	输入	8	通用输入 12	输入
9	通用输入 13	输入	10	通用输入 14	输入
11	通用输入 15	输入	12	-----------	--------
13	通用输入 11	输入	14	通用输入 10	输入
15	通用输入 9	输入	16	-----------	--------
17	通用输出 8	输出	18	通用输出 6	输出
19	通用输出 4	输出	20	通用输出 2	输出
21	通用输入 8	输入	22	通用输入 7	输入
23	通用输入 2	输入	24	通用输入 6	输入
25	通用输入 4	输入	26	通用输入 3	输入
27	通用输出 15	输出	28	通用输出 16	输出
29	通用输出 13	输出	30	通用输出 14	输出
31	通用输出 11	输出	32	通用输出 12	输出
33	通用输出 9	输出	34	通用输出 10	输出
35	- - - - - -	- - - -	36	- - - - - -	- - -
37	通用输出 7	输出	38	通用输出 5	输出
39	通用输出 3	输出	40	通用输出 1	输出

3.2 接线方法

MPC08 卡的开关量输入信号（限位、减速、原点、外部报警和通用输入），可以是触点型开关，也可以是 NPN 输出的传感器接近开关等。其接线方法如图 3.1 所示。

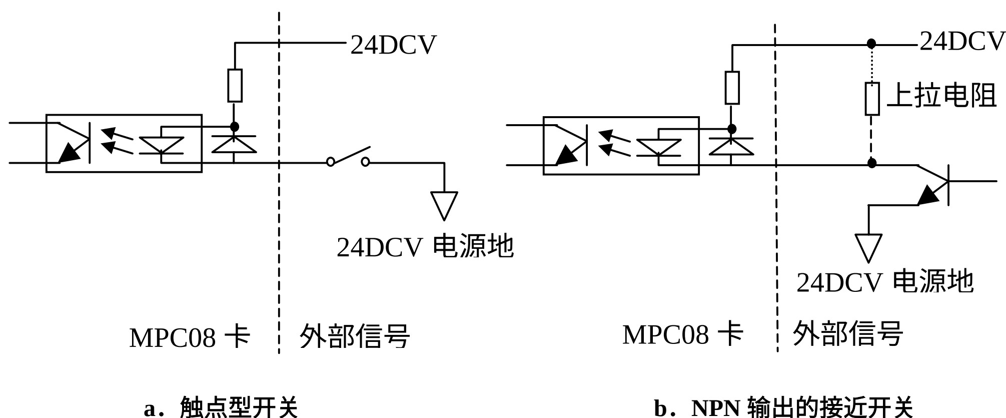
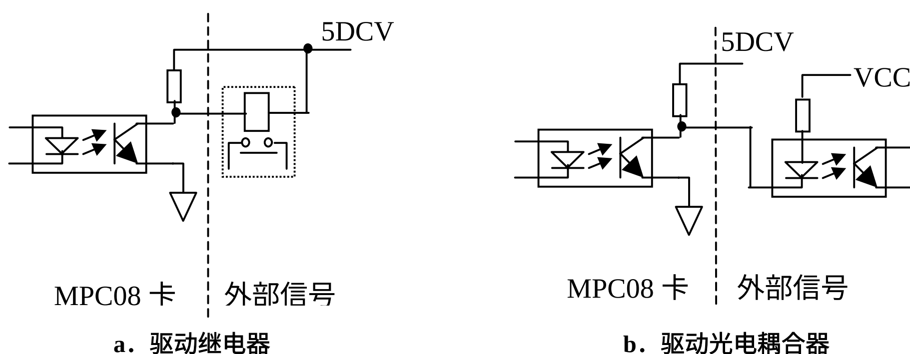


图 3.1　开关量输入信号接线方法

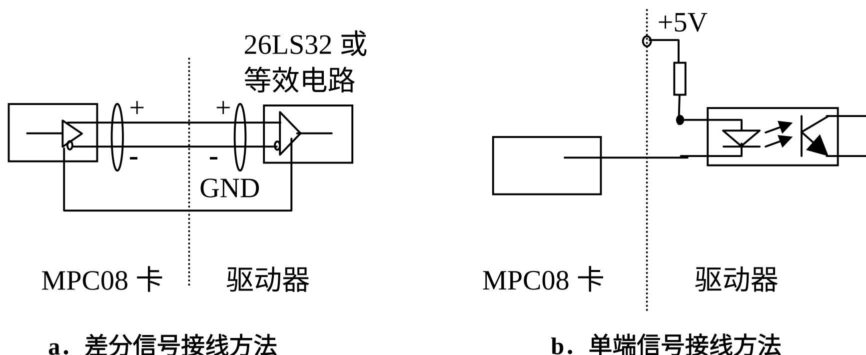
MPC08 卡的开关量输出信号，可以作为伺服系统的输入开关信号（伺服-ON、偏差计数器清零）或驱动 12/24DCV 的中间继电器（**需增加功率驱动电路**）或光电耦合器。其接线方法如图 3.2 所示。



a．驱动继电器　　**b．驱动光电耦合器**

图 3.2　开关量输出信号接线方法

MPC08 卡的脉冲/方向输出信号，作为步进电机或数字式伺服电机驱动器的控制信号，脉冲信号的频率决定电机的转速，脉冲信号的个数决定电机的转角。脉冲和方向信号的接线方法如图 3.3 所示。



a．差分信号接线方法　　**b．单端信号接线方法**

图 3.3　脉冲/方向输出信号接线方法

3.3 跳线说明

MPC08 通用 IO 扩展板跳线位置示意图如 3.4 所示。

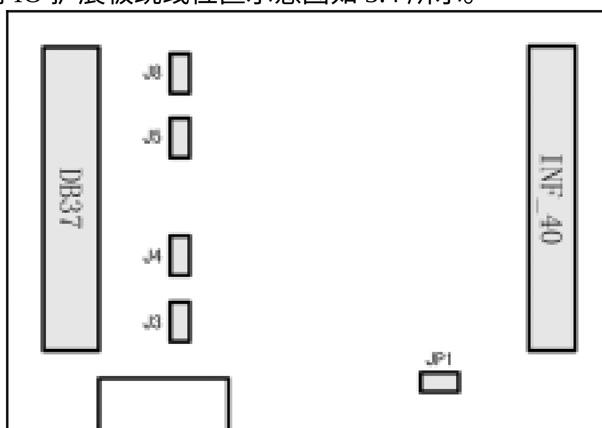


图 3.4　扩展板跳线位置示意图

各跳线连接方式如表 3-7 所示，使用前请检查。

表 3-7　扩展板连接器和跳线的定义

名称	使用状态	备注
JP1	连接 1、2 脚	由 PCI 总线给扩展板供电
	连接 2、3 脚	由 40 芯扁平电缆给扩展板供电
J3	断开	
J4	连接	输入电源：5V
J5	连接	输出电源：5V
J8	断开	
INF_40		40 芯扁平电缆接线端
DB62		DB62 接线端

使用扩展板跳线需注意：

1．供用户使用的跳线为：JP1、J3、J4、J5、J8

2．J3、J4 只能同时选择一个连接，默认状态 J4 连接、J3 断开，表示输入信号由外部提供的 5DCV 开关电源驱动。若 J3 连接、J4 断开，表示输入信号由外部提供的 24DCV 开关电源驱动。

3．J5、J8 只能同时选择一个连接，默认状态 J5 连接、J8 断开，表示输出信号由外部提供的 5DCV 开关电源驱动。若 J8 连接、J5 断开，表示输出信号由外部提供的 24DCV 开关电源驱动。

注意：

- 必须由外部提供脉冲、方向信号的 5DCV 开关电源；
- 若要使用输入开关量信号，则必须由外部提供 12~24DCV 的开关电源。

4 运动控制系统的开发

4.1 开发 Windows 下的运动控制系统

利用 MPC08 的动态链接库（DLL），开发者可以很快开发出 Windows 平台下的运动控制系统。MPC08 动态链接库是标准的 Windows 32 位动态链接库，选用的开发工具应支持 Windows 标准的 32 位 DLL 调用。

以下介绍如何利用两种常用的开发工具 Microsoft Visual Basic 和 Microsoft Visual C++开发基于 Windows 平台的运动控制程序。

4.1.1 开发 Visual Basic 控制程序

（一） 概述

为了开发基于 Windows 的运动控制程序，用户可以使用 VB5.0 或更高版本，开发一个简单的 Visual Basic 控制程序非常容易。按照如下步骤可以快速开发一个简单的控制程序。

1．安装 MPC08 驱动程序及函数库；
2．用 Visual Basic 写一个界面程序；
3．将 MPC08.bas 文件添加到 VB Project 中去；
4．在应用程序中调用运动函数。

所有 Visual Basic 的教材都介绍了如何写介面程序，包括按钮、对话框以及菜单等。对于熟悉 Visual Basic 和 MPC08 运动函数库的开发者来说，一个由输入框和命令按钮组成的基于 Windows 的简单运动程序，可以在几分钟内就可以开发出来。当然，开发一个复杂的运动应用程序所需的时间可能会长一些。

（二） 动态链接库函数调用方法

在 VB 中调用动态链接库（DLL）中函数应包括两部分工作：

● 函数声明

每一个动态链接库（DLL）中的函数在 VB 中的声明已经包含在 MPC08.bas 文件中了，该文件可在 MPC08 卡软件安装盘\Develop\VB 文件夹下找到，用户只需要将该文件添加进 VB 工程中即可。

● 函数调用

若调用函数的返回值为空或不需要返回值，则按如下方法调用：

con_pmove1,2000

或

Call con_pmove (1,2000)

若要得到函数的返回值，则按如下方法调用：

```
Dim rtn As Long
rtn=con_pmove(1,2000)
```

注意：传递的参数数据类型及接收返回值的变量类型应与函数声明的数据类型一致，并且建议函数描述中所有 int 型（C 语言中的整形）和 long 型（C 语言中的长整形）参数及返回值均统一采用 Long 型（VB 中的长整形）数据类型；所有的 float（C 语言中的单精度浮点型）和 double（C 语言中的双精度浮点型）参数及返回值均统一采用 Double 型（VB 中的双精度浮点型）数据类型，否则将可能产生无法预料的结果。

（三）演示示例程序的使用

MPC08 卡软件安装盘\Demo\VBDemo 文件夹下有两个在 VB6.0 下开发的运动控制系统演示示例程序。用户可按照如下步骤编译并运行该示例，在熟悉了相应编程方法后，用户可根据需要开发自己的运动控制系统。

（1） 按照 MPC08 软件的安装步骤进行正确安装。
（2） 安装完成后，可在安装盘\Demo\VBDemo\文件夹中下找到 Demo1 或 Demo2 文件夹。
（3） 启动 VB6.0 集成环境，并打开工程。
（4） 确保板卡已经正确设置并插入到计算机中。
（5） 编译该工程生成 EXE 文件。
（6） 运行生成的 EXE 文件。

4.1.2 用 Visual C++开发控制程序

（一）开发环境

用户可以使用 VC5.0 或更高版本，来进行 Windows 平台下运动控制系统开发。

（二）动态链接库函数调用方法

在 VC 中调用动态链接库 DLL 中函数有两种方法：

- 隐式调用

隐式调用需要如下文件：

（1） DLL 函数声明头文件 MPC.h；
（2） 编译连接时用的导入库文件 MPC08.lib；
（3） 动态链接库文件 MPC08.dll；
（4） 设备驱动程序 MPC08.sys；

以上文件中的（1）（2）可在 MPC08 卡软件安装盘\Develop\VC 文件夹下找到。（3）则已经由安装程序安装到 C:\WINDOWS\SYSTEM 文件夹下。（4）已经由安装程序安装到 C:\WINDOWS\SYSTEM32\DRIVERS 文件夹下（假定 Windows 安装在 C:\WINDOWS 文件夹下）。

建立工程之后，在 VC 集成环境中点击“ /project/settings… ”菜单弹出“ project settings ”对话框。选“ Link ”选项卡，在“ object/library modules ”栏内输入导入库文件名 MPC08.lib，单击“ OK ”按钮。在调用 DLL 函数的源代码文件开始处包含 MPC.h 头文件。之后则可以按照调用内部函数一样调用 DLL 函数。具体可参见演示示例：\Demo\VCDemo\Demo1。

- 显式调用

显式调用只需要如下文件：

（1） 动态链接库文件 MPC08.dll；

（2） 设备驱动程序 MPC08.sys。

以上文件中（1）已经由安装程序安装到 C:\WINDOWS\SYSTEM 文件夹下，（2）已经由安装程序安装到 C:\WINDOWS\SYSTEM32\DRIVERS 文件夹下（假定 Windows 安装在 C:\WINDOWS 文件夹下）。

显式调用方法需要调用 Windows API 函数加载和释放动态链接库。方法如下：

（1） 调用 Windows API 函数 LoadLibrary()动态加载 DLL；

（2） 调用 Windows API 函数 GetProcAddress()取得将要调用的 DLL 中函数的指针；

（3） 用函数指针调用 DLL 中函数完成相应功能；

（4） 在程序结束时或不再使用 DLL 中函数时，调用 Windows API 函数 FreeLibrary()释放动态链接库。

该方法比较烦琐。MPC08 软件中已经将常用的 MPC08.dll 中 DLL 函数封装成类 CLoadDll，并提供该类的源代码。该类含有与运动指令库函数名及参数相同的成员函数。源代码可在 MPC08 卡软件安装盘\Develop\VC 文件夹下找到，文件名为 LoadDll.cpp 和 LoadDll.h。开发人员可将其添加进工程，在程序适当地方添加该类的对象，通过对应成员函数来调用 DLL 中的函数。具体可参见演示示例：\Demo\VCDemo\Demo2。

以上在两种方法均为 VC 中调用动态链接库函数的标准方法，若要获得更具体的调用方法和帮助，请参考微软 Visual Stutio 开发文档 MSDN 或相关 VC 参考书籍中相应部分内容。

（三）演示示例程序的使用

MPC08 卡软件安装盘\Demo\VCDemo\文件夹下有三个在 VC6.0 下开发的运动控制系统演示示例程序。\Demo\VCDemo\Demo1 为隐式调用示例；\Demo\VCDemo\Demo2 为显式调用示例。用户可按照如下步骤编译并运行示例，在熟悉了相应编程方法后，用户可根据需要开发自己的运动控制系统。

（1） 按照 MPC08 软件的安装步骤进行正确安装。
（2） 安装完成后，可在安装盘\Demo\VBDemo\文件夹中下找到 Demo1 或 Demo2 文件夹。
（3） 启动 VC6.0 集成环境，并打开工程 demo1.dsw 或 demo2.dsw。
（4） 确保板卡已经正确设置并插入到计算机中。
（5） 编译连接该工程生成 EXE 文件。
（6） 运行生成的 EXE 文件。

另外，在\Demo\VCDemo\文件夹下还提供了一个 MPC08 函数测试程序\Demo\VCDemo\Demo3，只提供了可执行文件，可测试 MPC08 所有函数。

5 函数描述

本章详细地描述了 MPC08 运动库中的每一个函数。其中，在函数库中使用的单位和函数返回值约定通常如下：

单位

- 位移（或距离）的单位为 P（Pulse），即脉冲数；
- 速度的单位是 PPS（Pulse/sec），即脉冲/秒；
- 加速度和减速度的单位是 PPSS（Pulse/sec^2），即脉冲／秒 ^{2}。

函数返回值

运动库中的大多数函数是整型函数。一般情况下，如不作特殊说明，它们的返回值意义为：

0　　函数执行正确；

-1　　函数执行错误。

5.1 控制卡和轴设置函数

该类函数主要用于设置 MPC08 卡的使用数量、控制轴数以及每轴的输出模式，速度、加速度等的设置和读取等等。相关函数有 15 个：

int auto_set(void)；/*自动检测和自动设置控制卡*/

int init_board（void）；/*对控制卡硬件和软件初始化*/

int get_max_axe（void）；/*读取总轴数*/

int get_board_num（void）；/*读取板卡数*/

int get_axe（int board_no)；/*读取板卡上轴数*/

int set_outmode（int ch，int mode，int outlogic）；/*设置各轴输出模式*/

int set_home_mode（int ch，int home_mode）；/*设置回原点模式*/

int set_conspeed（int ch，double conspeed）；/*设置各轴常速度*/

double get_conspeed（int ch）；/*读取各轴常速度*/

int set_profile（int ch，double ls，double hs，double acc）；/*设置梯形速度*/

int get_profile（int ch，double &ls，double &hs，double &acc）；/*读取各轴梯形速度*/

int set_vector_conspeed（int con_speed）；/*设置常矢量速度*/

int set_vector_profile（double vec_fl，double vec_fh，double vec_ad）；
/*设置梯形矢量速度*/

double get_vector_conspeed（void）；/*读取矢量常速度*/

int get_vector_profile（double *vec_fl，double *vec_fh，double *vec_ad）；
/*读取矢量梯形速度*/

double get_rate（int ch）；/*读取当前运转速度*/

函 数 名：auto_set

目　　的：用 auto_set 函数自动检测 MPC08 卡的数量、各卡上的轴数，并自动设置每块 MPC08 控制卡。

语　　法：int auto_set（void）；

调用例子：auto_set（）；/*自动检测和自动设置 MPC08 卡*/
描　　述：可以调用 auto_set 完成 MPC08 卡的数量、轴数的自动检测，并自动设置这些参数。该函数在程序中只能调用一次。
返 回 值：如果调用成功，auto_set 函数返回总轴数；若检测不到卡，返回 0；调用失败返回负数。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
注　　释：
参　　见：

函 数 名：**init_board**
目　　的：用 init_board 函数初始化控制卡。
语　　法：int init_board（void）；
调用例子：init_board（）；
描　　述：在用 auto_set 自动检测和设置之后，必须调用 init_board 函数来对控制卡进行初始化。init_board 函数主要初始化控制卡的各个寄存器、各轴的脉冲输出模式（脉冲/方向）、常速度（2000pps）、梯形速度（初速 2000pps，高速 8000pps，加减速 80000ppss）、矢量常速度（2000pps）、矢量梯形速度（初速 2000pps，高速 8000pps，加减速 80000ppss）等等。该函数在程序中只能调用一次。
返 回 值：如果调用成功，init_board 函数返回插入的板卡数；若检测不到卡，返回 0；负数表示出错。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
注　　释：如果不调用 init_board 函数初始化，控制卡将不能正常工作。若需改变脉冲输出模式、速度等初始化数据，可调用其它函数来修改。
参　　见：auto_set

函 数 名：**get_max_axe**
目　　的：get_max_axe 用于读取总的控制轴数。
语　　法： int get_max_axe（void）；
调用例子：max_axe_num=get_max_axe（）；
返 回 值：get_max_axe 返回总控制轴数。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：**get_board_num**
目　　的：get_board_num 用于读取安装的 MPC08 板卡数。
语　　法： int get_board_num（void）；
调用例子：card_num=get_board_num（）；
返 回 值：get_board_num 返回总板卡数。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：**get_axe**
目　　的：get_axe 用于读取板卡上的轴数。

语　　法：int get_axe（int board_no）；
调用例子：axe_num=get_axe（1）；
返 回 值：get_axe 返回板卡上的轴数。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：set_outmode
目　　的：用于设置每个轴的脉冲输出模式。如果驱动器要求双脉冲（正向脉冲、反向脉冲）控制信号接口，那么应在 init_board 函数后调用该函数。
语　　法：int set_outmode（int ch，int mode，int outlogic）；
ch：所设置输出方式的控制轴；
mode：脉冲输出模式设置（1 为脉冲／方向方式，0 为双脉冲方式）；
outlogic：该参数在 MPC08 中无效。
调用例子：set_outmode（2，0，1）；/*将第 2 轴的脉冲输出模式设置为双脉冲模式。*/
描　　述：在缺省情况下，init_board 函数将所有轴设置为脉冲／方向模式，输出信号为负逻辑。如果驱动器要求双脉冲（正向脉冲和反向脉冲）模式的输入，那么应在 init_board 函数后调用 set_outmode 重新设置所要求的模式。注意：控制卡的输出模式应与所连接的驱动器的输入信号模式一致，否则电机将不能正常工作。
返 回 值：如果输出方式设置成功，则 set_outmode 返回值为 0，否则返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：init_board

函 数 名：set_home_mode
目　　的：用于设置各轴回原点运动时检测原点信号的方式。
语　　法：int set_home_mode（int ch，int home_mode）
ch：是所设置的轴；
home_mode：回原点运动时检测原点信号的方式（0：仅检测原点接近开关信号，1：检测原点接近开关信号和电机上光电编码器 Z 相脉冲信号同时出现）。
调用例子：set_home_mode（1，1）；
描　　述：在被控设备（比如数控机床等）回原点运动时，MPC08 卡将自动检测原点信号，并在到达原点位置时自动停止运动。原点信号一般由接近开关发送。但在一些回原点时定位精度要求较高的场合，原点信号除了接近开关信号之外，还要检测执行电机上光电编码器的 Z 相脉冲，即仅当接近开关信号和 Z 相脉冲信号同时出现时，才表明已到达原点。函数 set_home_mode 就是用于设置每个轴在回原点运动时检测原点信号的方式。当 home_mode=0 时，仅将原点接近开关信号作为原点信号；当 home_mode=1 时，将原点接近开关信号和 Z 相脉冲信号两者同时出现作为原点信号。注意：只有执行电机上装有光电编码器时，才能将原点接近开关信号和 Z 相脉冲信号同时出现设置为回原点运动的检测信号，否则将无法正确完成回原点运动。
返 回 值：如果设置成功，返回 0，否则返回-1。

系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：set_maxspeed
目　　的：用于设置每个轴的最大速度。
语　　法：int set_maxspeed（int ch，double speed）；
　　　　　ch：所设置的控制轴；
　　　　　speed：设置的最大速度值，单位为脉冲／秒（pps）。；
调用例子：set_maxspeed（2，10000）；/*将第 2 轴的最大速度设置为 10000pps。*/
描　　述：在缺省情况下，init_board 函数将所有轴设置其最大速度，即 8200000。使用时可按照实际输出速度进行设置以获得比较好的速度精度。
返 回 值：如果输出方式设置成功，则 set_ maxspeed 返回值为 0，否则返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：set_conspeed

函 数 名：set_conspeed，get_conspeed
目　　的：用 set_conspeed 函数来设置一个轴在常速运动时的速度。
　　　　　用 get_conspeed 函数来获取某个轴所设置的常速度。
语　　法：int set_conspeed（int ch，double conspeed）；
　　　　　double get_conspeed（int ch）；
　　　　　ch：函数所操作的轴；
　　　　　conspeed：设定的常速度值，单位为脉冲／秒（pps）。
调用例子：set_conspeed（2，400）；
　　　　　speed=get_conspeed（2）；
描　　述：函数 set_conspeed 可以设定在常速运动方式下的速度。如果多次调用这个函数，最后一次设定的值有效，而且在下一次改变之前，一直保持有效。
返 回 值：如果常速度值设置成功，set_conspeed 返回 0 值，出错时返回-1。
　　　　　函数 get_conspeed 返回指定轴的常速度值，出错时返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
注　　释：常速度值一般设置较低，以免造成控制电机（尤其是开环的步进电机）丢步或过冲。如果需要高速运动，最好使用梯形速度方式。
参　　见：set_profile，set_vector_conspeed

函 数 名：set_profile，get_profile
目　　的：用 set_profile 函数来设定在快速运动（包括 fast_hmove，fast_vmove，fast_pmove 等）方式下的梯形速度的各参数值；用 get_profile 来读取梯形速度的各参数值。
语　　法：int set_profile（int ch，double ls，double hs，double accel）；
　　　　　ch：设定的轴数；
　　　　　ls：设定低速（起始速度）的值；单位为 pps（脉冲／秒）；
　　　　　hs：设定高速（恒速段）的速度值；单位为 pps（脉冲秒）；
　　　　　accel：设定加速度大小；单位为 ppss（脉冲／秒／秒）；
　　　　　int get_profile（int ch，int *ls，int *hs，long*accel）

double *ls:指定起始速度的指针；
double *hs:指向高速的指针；
double *accel：指向加速度值的指针。
调用例子：set_profile（3，600，6000，10000）；
get_profile，（3，&ls，&hs，&accel）；
描　　述：函数 set_profile 设定一个轴在快速运动方式下的低速（起始速度）、高速、加／减速度值（减速度值等于加速度值）。这几个参数的缺省值分别为 2000、8000、80000。函数 get_profile 通过指针返回一个轴的梯形速度的低速、高速和加／减速度值。
返 回 值：如果设定参数值成功，set_profile 返回 0，出错返回负数。
如果调用成功，get_profile 返回 0 值，否则返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：set_conspeed，set_vector_conspeed，set_vector_profile

函 数 名：set_vector_conspeed，get_vector_conspeed
目　　的：用 set_vector_conspeed 函数来设置常速方式下的矢量速度，这个矢量速度在两轴或三轴直线插补运动中将会用到；用 get_vector_conspeed 函数来读取常速方式下的矢量速度。
语　　法：int set_vector_conspeed（double vec_conspeed）；
vec_conspeed：在常速插补期间的矢量速度；
double get_vector_conspeed（void）；
调用例子：set_vector_conspeed（1000）；
vec_conspeed= get_vector_conspeed（）；
描　　述：函数 set_vector_conspeed 为下列二轴或三轴插补运动函数设置矢量速度：con_line2、con_line3 等。它不能为 fast_lin2、 fast_line3 等高速插补运动设置运动速度（它们的速度依赖于 set_vector_profile）。函数 get_vector_conspeed 返回常矢量速度。最后一次调用 set_vector_conspeed 的常矢量速度有效。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
注　　释：常矢量速度应设置为相对较小一些，以免在运动过程中丢步。对于高速运动插补，如：fast_line2、fast_line3 等来说，可用 set_vector_profile 来设置运动速度。
参　　见：set_vector_profile ，set_conspeed，set_profile

函 数 名：set_vector_profile，get_vector_profile
目　　的：用 set_vector_profile 来设置矢量梯形速度参数；
用 get_vector_profile 来获取矢量梯形速度参数值；
语　　法：int set_vector_profile（double vec_fl，double vec_fh，double vec_ad）；
vec_fl：矢量低速的速度值；
vec_fh：矢量高速的速度值；
vec_ad：矢量高速的加速度值；
int get_vector_profile（double *vec_fl，double *vec_fh，double *vec_ad）；
*vec_fl:指向矢量低速的指针；
*vec_fh:指向矢量高速的指针；

*vec_ad：指向矢量加速度的指针。
调用例子：set_vector_profile（1000，16000，10000）；
get_vector_profile（&vec_fl，&vec_fh，&vec_ad）；
描　　述：函数 set_vector_profile 为 fast_line2，fast_line3 等函数设置矢量梯形速度。这个函数不为 con_line2，con_line3 等函数设置运动速度。
返 回 值：如果调用成功，set_vector_profile 和 get_vector_profile 函数返回 0，在出错的情况下，返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
注　　释：
参　　见：set_vector_conspeed，fast_line2，fast_line3

函 数 名：get_rate
目　　的：用 get_rate 函数来获取当前正在运行的某个轴的速度。
语　　法：double get_rate（int ch）；
ch：函数所操作的轴；
调用例子：speed=get_rate（2）；
描　　述：函数 get_rate 可以在被控轴运动过程中读取该轴当前的运行速度。
返 回 值：函数 get_rate 返回指定轴的当前运行速度，单位：每秒脉冲数（pps），函数调用出错返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：get_profile

5.2 运动指令函数

按运动类型分类，主要有三种类型：点位运动、连续运动和回原点运动；按运动方式可分为独立运动和插补运动两种；按运动速度可分为常速运动和梯形速度两种。相关函数共有 22 个。为了描述方便，下面将运动指令分为独立运动和插补运动两部分来说明。

5.2.1 独立运动函数

所谓独立运动指各轴的运动之间没有联动关系，可以是单轴运动，也可以是多轴同时按各自的速度运动。点位运动、连续运动和回原点运动都属于独立运动。

独立运动指令的函数名格式为：X_YmoveZ

其中：

X：由 con 和 fast 替代，con 表示常速运动，fast 表示快速运动；
Y：由 p、v 和 h 替代，p 表示点位运动，v 表示连续运动，h 表示回原点运动；
move：为指令主体，表示该指令为运动指令；
Z：没有时为单轴运动，为 2 时表示两轴独立运动，为 3 时表示三轴独立运动。

例如：con_vmove 为单轴的常速连续运动函数；con_pmove2 为两轴的常速点位运动函数；fast_hmove3 为三轴的快速回原点运动指令。

对于常速运动指令，运动速度由 set_conspeed 设定；对于快速运动指令，运动

速度由 set_profile 设定。

下面以点位运动、连续运动和回原点运动分别说明各运动指令的含义。

一、点位运动函数

点位运动是指被控轴以各自的速度分别移动指定的距离，在到达目标位置时自动停止。注意：在两轴或三轴的点位运动函数中，各轴同时开始运动，但不一定同时到达目标位置。在 MPC08 函数库中共提供了六个点位运动指令函数：

int con_pmove（int ch，long step）；/*一个轴以常速做点位运动*/

int fast_pmove（int ch，long step）；/*一个轴以快速做点位运动*/

int con_pmove2（int ch1，long step1，int ch2，long step2）；/*两轴以常速做点位运动*/

int fast_pmove2（int ch1，long step1，int ch2，long step2）；/*两轴以快速做点位运动*/

int con_pmove3（int ch1，long step1，int ch2，long step2，int ch3，long step3）；
/*三个轴以常速做点位运动*/

int fast_pmove3（int ch1，long step1，int ch2，long step2，int ch3，long step3）；
/*三个轴以快速作点位运动*/

其中：

ch、ch1、ch2、ch3：被控轴的轴号；

step、step1、step2、step3：表示被控轴从当前位置开始移动的距离，正数表示正方向；负数表示负方向，其单位为脉冲数。

调用例子：

con_pmove（1，-2000）；/*第一轴以其常速向负方向移动 2000 个脉冲的距离*/

fast_pmove2（2，5000，3，-1000）；/*第二轴以快速向正方向移动 5000 个脉冲的距离；第三轴以快速向负方向移动 1000 个脉冲的距离。*/

返回值：如果调用成功，这些函数返回 0，在出错情况下返回-1。

二、连续运动函数

连续运动是指被控轴以各自的速度按给定的方向一直运动，直到碰到限位开关或调用制动函数才会停止。在 MPC08 函数库中共提供了六个连续运动指令函数：

int con_vmove（int ch，int dir）；/*一轴以常速连续运动*/

int fast_vmove（int ch，int dir）；/*一轴以快速连续运动*/

int con_vmove2（int ch1，int dir1，int ch2，int dir2）；/*两轴以常速连续运动*/

int fast_vmove2（int ch1，int dir1，int ch2，int dir2）；/*两轴以快速连续运动*/

int con_vmove3（int ch1，int dir1，int ch2，int dir2，int ch3，int dir3）；
/*三个轴以常速连续运动*/

int fast_vmove3（int ch1，int dir1，int ch2，int dir2，int ch3，int dir3）；
/*三个轴以快速连续运动*/

其中：

ch、ch1、ch2、ch3：被控轴的轴号；

dir、dir1、dir2、dir3：表示被控轴的运动方向，+1 表示正方向；-1 表示负方向。

调用例子：

con_vmove（1，-1）；/*第一轴以其常速向负方向连续运动*/

fast_vmove2（2，1，3，-1）；/*第二轴快速向正方向连续运动；第三轴快速向负方向连续运动。*/

返回值：如果调用成功，这些函数返回 0，在出错情况下返回-1。

三、回原点函数

回原点运动是指被控轴以各自的速度按给定的方向一直运动，直到碰到原点信号、限位开关或调用制动函数才会停止。在 MPC08 函数库中共提供了六个回原点运动指令函数：

int con_hmove（int ch，int dir）；/*以常速返回原点*/

int fast_hmove（int ch，int dir）；/*以快速返回原点*/

int con_hmove2（int ch1，int dir1，int ch2，int dir2）；/*两轴以常速各自返回原点*/

int fast_hmove2（int ch1，int dir1，int ch2，int dir2）；/*两轴以快速各自返回原点*/

int con_hmove3（int ch1，int dir1，int ch2，int dir2，int ch3，int dir3）；
/*三个轴以常速各自返回原点*/

int fast_hmove3（int ch1，int dir1，int ch2，int dir2，int ch3，int dir3）；
/*三个轴以快速各自返回原点*/

其中：

ch、ch1、ch2、ch3：被控轴的轴号；

dir、dir1、dir2、dir3：表示被控轴的运动方向，+1 表示正方向；-1 表示负方向。

调用例子：

con_hmove（1，-1）；/*第一轴以其常速向负方向作回原点运动*/

fast_hmove2（2，1，3，-1）；/*第二轴快速向正方向作回原点运动；第三轴快速向负方向作回原点运动。*/

返回值：如果调用成功，这些函数返回 0，在出错情况下返回-1。

注释：要成功地实现回原点运动，运动轴上应设有常开型原点开关（接近开关或传感器），低电平或下降沿有效。

5.2.2 插补运动函数

插补运动是指两轴或三轴按照一定的算法进行联动，被控轴同时启动，并同时到达目标位置。插补运动以矢量速度运行，矢量速度分为常矢量速度和梯形矢量速度。与插补运动有关的函数有：

一、线性插补函数

线性插补运动是指两个轴或三个轴以矢量速度（常矢量速度或梯形矢量速度）作线性联动，每个被控轴的运动速度为矢量速度在该轴上的分速度，各个被控轴同时启动，并同时到达目标位置。MPC08 函数库中提供四个线性插补函数：

int con_line2（int ch1，long pos1，int ch2，long pos2）；
/*两轴做常速直线运动*/

int fast_line2（int ch1，long pos1，int ch2，long pos3）；
/*两轴做快速直线运动*/

int con_line3（int ch1，long pos1，int ch2，long pos2，int ch3，long pos3）；

/*三个轴直线运动*/

int fast_line3（int ch1，long pos1，int ch2，long pos2，int ch3，long pos3）；

/*三个轴做快速直线运动*/

其中：

ch1、ch2、ch3：被控轴的轴号；

pos1、pos2、pos3：表示被控轴从当前位置开始移动的距离，正数表示正方向；负数表示负方向，其单位为脉冲数。

调用例子：

con_line2（1，-2000，3，1000）；

/*第一轴和第三轴以常矢量速度作线性插补运动，第一轴向负方向移动 2000 个脉冲的距离，同时第三轴向正向移动 1000 个脉冲的距离*/

fast_line3（2，5000，3，-1000，5，3000）；

/*第二轴、第三轴和第五轴以梯形矢量速度作线性插补运动，第二轴向正方向移动 5000 个脉冲的距离；第三轴向负方向移动 1000 个脉冲的距离；第五轴向正方向移动 3000 个脉冲的距离。*/

返回值：如果调用成功，这些函数返回 0，在出错情况下返回-1。

5.3 制动函数

在运动过程中，如果需要暂停或中止某个轴或某几个轴的运动，可以调用制动函数来完成。在 MPC08 运动函数库中提供了 6 个制动函数：

void sudden_stop（int ch）；*/立即制动一个运动轴*/

void sudden_stop2（int ch1，int ch2）；*/立即制动两个运动轴*/

void sudden_stop3（int ch1，int ch2，int ch3）；*/立即制动三个运动轴*/

void decel_stop（int ch）；/*光滑制动一个运动轴*/

void decel_stop2（int ch1，int ch2）；/*光滑制动两个运动轴*/

void decel_stop3（int ch1，int ch2，int ch3）；/*光滑制动三个运动轴*/

其中：

ch、ch1、ch2、ch3：被控轴的轴号；

调用例子：

decel_stop（2）；/*光滑制动第二轴*/

sudden _stop2（1，4）；/*立即制动第一、四轴*/

decel_stop3（1，2，3）；/*光滑制动第一、二、三轴*/

返回值：调用成功返回 0，否则返回-1 或未能制动的轴号。

说明：制动函数对所有类型的运动都有效。decel_stop 类型的制动函数一般用于梯形速度运动方式（fast_YmoveZ），它可以使被控轴的速度先从高速降至低速（由 set_proflile 设定），然后停止运动。一般在运动过程需要暂停时应调用 decel 类型制动函数，以便能够光滑地中止快速运动（如：fast_hmove、fast_vmove、fast_pmove2 等），以免发生过冲现象。sudden_stop 类型制动函数使被控轴立即中止运动，这个函数执行后，控制卡立即停止向电机驱动器发送脉冲，使之停止运动。该函数通常在紧急停车时调用。对于常速运动方式（con_YmoveZ），这两类制动函数效果一样。

5.4 位置和状态设置函数

在 MPC08 运动函数库中提供了 9 个位置和状态设置函数：

int set_abs_pos (int ch，long pos) ；/*设置一个轴的绝对位置值*/
int reset_pos（int ch）；/*当前位置值复位至零*/
int reset_cmd_counter()；/*用于对运动指令计数清零*/
int set_getpos_mode(int ch,int mode);/*设置 get_encoder( )函数返回值的来源*/
int set_encoder_mode(long ch,long mode,long multip,long count_unit);
/*设置一个轴的编码器反馈信号模式*/
int set_dir(int ch，int dir)；/*设置一个轴的运动方向*/
int enable_sd(int ch，int flag)；/*设置一个轴的外部减速信号是否有效*/
int enable_el(int ch，int flag)；/*设置一个轴的外部限位信号是否有效*/
int enable_org(int ch，int flag)；/*设置一个轴的外部原点信号是否有效*/
int set_sd_logic（int ch，int flag）/*用于设置轴的外部减速信号有效电平*/
int set_el_logic（int ch，int flag）/*用于设置轴的外部限位信号有效电平*/
int set_org_logic（int ch，int flag）/*用于设置轴的外部原点信号有效电平*/
int set_alm_logic（int ch，int flag）/*用于设置轴的外部报警信号有效电平*/

函 数 名：set_abs_pos
目　　的：用于设置轴的运动起始绝对位置。
语　　法：int set_abs_pos（int ch，long pos）；
　　　　　ch：所要设置的轴；
　　　　　pos：所要设置的该轴的起始绝对位置；
调用例子：set_abs_pos（1，1000）；/*将第 1 轴的当前位置设置为 1000*/
描　　述：调用该函数可将当前绝对位置设置为某一个值，但从上一个位置到该位置之间不会产生轴的实际运动。调用该函数并将第二个参数设为 0 可实现 reset_pos()函数的功能。
返 回 值：如果函数调用成功，则返回值为 0；否则若返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：reset_pos
语　　法：int reset_pos（int ch）；
　　　　　ch：被复位轴的轴号；
调用例子：int reset_pos（1）；
描　　述：函数 reset_pos 将指定轴的绝对位置和相对位置复位至 0，通常在轴的原点找到时调用，调用这个函数后，当前位置值变为 0，这以后，所有的绝对位置值均是相对于这一点的。调用该函数时必须确保该轴运动已经停止，否则将引起绝对位置值的混乱。
返 回 值：如果调用成功，reset_pos 返回 0，否则返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
注　　释：一般来说，这个函数应在成功地执行 con_hmove 或 fast_hmove 后调用。

参　　见：get_abs_pos，get_rel_pos

函 数 名：reset_cmd_counter
目　　的：用于对运动指令计数清零。
语　　法：int reset_cmd_counter（）；
调用例子：reset_cmd_counter（）；
描　　述：运动指令计数从初始化完成后的 0 开始，随着执行的运动指令（即能产生运动的指令，不包括 set_conspeed 等设置指令）递增，可以调用 reset_cmd_counter()函数进行清零。
返 回 值：如果调用成功，则返回值为 0，否则返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：set_cmd_counter

函 数 名：set_getpos_mode
目　　的：用于设置调用 get_encoder()函数获取的位置值的来源。
语　　法：int set_getpos_mode(int ch,int mode)；
ch：所要设置的轴；
mode：调用 get_encoder()函数获取的位置值的来源（1 为编码器反馈信号，0 为输出脉冲数）；
调用例子：set_ getpos _mode（1，1）；/*将第 1 轴设定为编码器反馈信号*/
描　　述：缺省情况下调用 get_encoder()函数获取的位置值为实际输出脉冲数，若要让调用 get_encoder 函数获取的位置值为编码器反馈信号，则应在调用 init_board()函数之后调用该函数和 set_encoder_mode()函数进行设置，否则调用 get_encoder 函数获取的位置值将不是读取的编码器位置。调用该函数并将第二个参数设置为 1 后，编码器反馈信号模式将被设置为 A/B 相 90 度相位差方式，1 倍频，若要该为其他模式，可调用 set_encoder_mode()函数进行设置。
返 回 值：如果设置成功，则 set_ getpos _mode 返回值为 0，否则返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：get_encoder，set_encoder_mode

函 数 名：set_encoder_mode
目　　的：用于设置每个轴的编码器反馈信号模式。
语　　法：int set_encoder_mode（int ch，int mode，int multip，int count_unit）；
ch：所设置的控制轴；
mode：在 MPC07 中无用，系统始终默认为 A/B 相 90 度相位差方式。
multip：A/B 相 90 度相位差方式时的倍频：1、4。
count_unit：在 MPC07 中无用。
调用例子：set_encoder_mode（1，0，1，0）；/*将第 1 轴的编码器反馈信号设置为 A/B 相 90 度相位差 1 倍频方式。*/
描　　述：在缺省情况下，init_board 函数将所有轴设置为 A/B 相 90 度相位差 1 倍频方式。若要做其它设置，应在 init_board 函数后调用 set_encoder_mode 重新设置所要求的模式。

返 回 值：如果设置成功，则 set_encoder_mode 返回值为 0，否则返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：get_encoder

函 数 名：**set_dir**
目　　的：用于设置轴的运动方向。
语　　法：int set_dir（int ch，int dir）；
　　ch：所要设置的轴；
　　dir：表示被控轴的运动方向，+1 表示正方向；-1 表示负方向。；
调用例子：set_dir（1，-1）；/*将第 1 轴的运动方向设置成为负方向*/
描　　述：调用该函数可在新的运动指令发出前设定某轴的运动方向。
返 回 值：如果函数调用成功，则返回值为 0；否则若返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：**enable_sd**
目　　的：用于设置轴的外部减速信号是否有效。
语　　法：int enable_sd（int ch，int flag）；
　　ch：所要设置的轴；
　　flag：外部减速信号有无效的标志，1 表示外部减速信号有效；0 表示外部减速信号无效。
调用例子：enable_sd（1，0）；/*将第 1 轴的外部减速信号设置为无效*/
描　　述：调用该函数设置某轴的外部减速信号是否有效。如果将某轴的外部减速信号设置为无效，则对应的减速信号输入端口（SD）可作为通用输入口使用，使用函数 check_SFR 可读取相应端口状态。
返 回 值：如果函数调用成功，则返回值为 0；否则若返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：check_SFR

函 数 名：**enable_el**
目　　的：用于设置轴的外部限位信号是否有效。
语　　法：int enable_el（int ch，int flag）；
　　ch：所要设置的轴；
　　flag：外部限位信号有无效的标志，1 表示外部限位信号有效；0 表示外部限位信号无效。
调用例子：enable_el（1，0）；/*将第 1 轴的外部限位信号设置为无效*/
描　　述：调用该函数设置某轴的外部限位信号是否有效。如果将某轴的外部限位信号设置为无效，则对应的限位信号输入端口（EL+、EL-）可作为通用输入口使用，使用函数 check_SFR 可读取相应端口状态。

返 回 值：如果函数调用成功，则返回值为 0；否则若返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：check_SFR

函 数 名：enable_org
目　　的：用于设置轴的外部原点信号是否有效。
语　　法：int enable_org（int ch，int flag）；
　　ch：所要设置的轴；
　　flag：外部原点信号有无效的标志，1 表示外部原点信号有效；0 表示外部原点信号无效。
调用例子：enable_org（1，0）；/*将第 1 轴的外部原点信号设置为无效*/
描　　述：调用该函数设置某轴的外部原点信号是否有效。如果将某轴的外部原点信号设置为无效，则对应的原点信号输入端口（ORG）可作为通用输入口使用，使用函数 check_SFR 可读取相应端口状态。
返 回 值：如果函数调用成功，则返回值为 0；否则若返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：check_SFR

函 数 名：set_sd_logic
目　　的：用于设置轴的外部减速信号有效电平。
语　　法：int set_sd_logic（int ch，int flag）；
　　ch：所要设置的轴；
　　flag：外部减速信号有效电平标志，1 表示外部减速信号高电平有效；0 表示外部减速信号低电平有效。
调用例子：set_sd_logic（1，1）；/*将第 1 轴的外部减速信号设置为高电平有效*/
描　　述：调用该函数设置某轴的外部减速信号有效电平。如果将某轴的外部减速信号设置为高电平有效，则对应的减速信号输入端口为高电平时轴自动减速。初始化时系统默认低电平有效。
返 回 值：如果函数调用成功，则返回值为 0；否则若返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：set_el_logic
目　　的：用于设置轴的外部限位信号有效电平。
语　　法：int set_el_logic（int ch，int flag）；
　　ch：所要设置的轴；
　　flag：外部限位信号有效电平标志，1 表示外部限位信号高电平有效；0 表示外部限位信号低电平有效。
调用例子：set_el_logic（1，1）；/*将第 1 轴的外部限位信号设置为高电平有效*/

描　　述：调用该函数设置某轴的外部限位信号有效电平。如果将某轴的外部限位信号设置为高电平有效，则对应的限位信号输入端口为高电平时轴自动停止运动。初始化时系统默认低电平有效。

返 回 值：如果函数调用成功，则返回值为 0；否则若返回-1。

系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP

参　　见：

函 数 名：set_org_logic

目　　的：用于设置轴的外部原点信号有效电平。

语　　法：int set_org_logic（int ch，int flag）；

ch：所要设置的轴；

flag：外部原点信号有效电平标志，1 表示外部原点信号高电平有效；0 表示外部原点信号低电平有效。

调用例子：set_org_logic（1，1）；/*将第 1 轴的外部原点信号设置为高电平有效*/

描　　述：调用该函数设置某轴的外部原点信号有效电平。如果将某轴的外部原点信号设置为高电平有效，则对应的原点信号输入端口为高电平时表示轴回到原点。初始化时系统默认低电平有效。

返 回 值：如果函数调用成功，则返回值为 0；否则若返回-1。

系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP

参　　见：

函 数 名：set_alm_logic

目　　的：用于设置轴的外部报警信号有效电平。

语　　法：int set_alm_logic（int ch，int flag）；

ch：所要设置的轴；

flag：外部报警信号有效电平标志，1 表示外部报警信号高电平有效；0 表示外部报警信号低电平有效。

调用例子：set_alm_logic（1，1）；/*将第 1 轴的外部报警信号设置为高电平有效*/

描　　述：调用该函数设置某轴的外部报警信号有效电平。如果将某轴的外部报警信号设置为高电平有效，则对应的报警信号输入端口为高电平时轴自动停止运动。初始化时系统默认低电平有效。

返 回 值：如果函数调用成功，则返回值为 0；否则若返回-1。

系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP

参　　见：

5.5 位置和状态查询函数

在运动过程中，如果需要查询某个轴的运动位置或状态，可以调用位置或状态查询函数。在 MPC08 运动函数库中提供了 11 个位置和状态查询函数：

5.5.1 位置查询函数

int get_abs_pos（int ch，long*pos）；/*返回一个轴的绝对位置值*/
int get_rel_pos（int ch，long*pos）；/*返回一个轴的相对位置值*/
int get_encoder（int ch，long*pos）；/*返回一个轴的实际位置值*/
int get_cur_dir (int ch)；/*返回一个轴的当前运动方向*/

函 数 名：get_abs_pos，get_rel_pos，get_encoder
目　　的：用 get_abs_pos 读取一个相对于初始位置或原点位置的绝对位置。
用 get_rel_pos 读取一个相对于当前运动起始点的相对位置值。
用 get_encoder 读取一个相对于初始位置或原点位置的编码器反馈的实际位置值。
语　　法：int get_abs_pos（int ch，long*abs_pos）；
int get_rel_pos（int ch，long*rel_pos）；
int get_encoder（int ch，long*en_pos）；
ch：读取位置的轴号；
abs_pos：一个指向绝对位置的长整型指针；
rel_pos：一个指向相对位置的长整型指针；
en_pos：一个指向实际位置的长整型指针；
调用例子：temp=get_abs_pos（1，&abs_pos）；
temp=get_rel_pos（1，&rel_pos）；
描　　述：函数 get_abs_pos 获取指定轴的当前绝对位置，如果执行过回原点运动，那么这个绝对位置是相对于原点位置的；如果没有执行过回原点运动，那么这个绝对位置是相对于开机时的位置。函数 get_rel_pos 获取对应于当前运动起始点的相对位置值，如果指定轴当前没有运动，那么该轴的相对位置为 0。由于这两个函数读取的位置值是由控制卡输出脉冲的数量决定的，所以在丢步或过冲等情况下，不能反映实际的位置值。get_encoder 读取光电盘反馈的实际位置。
返 回 值：如果调用成功，get_abs_pos、get_rel_pos 和 get_encoder 返回 0 值，在出错情况下返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP

函 数 名：get_cur_dir
目　　的：用于获取轴的当前运动方向。
语　　法：int get_cur_dir（int ch）；
ch：所要查询的轴；
调用例子：get_cur_dir（1）；/*获取第 1 轴的当前运动方向*/
返 回 值：如果函数调用失败，则返回值为-2；否则若返回-1 表示当前运动方向负向，返回 1 则表示当前运动方向正向，0 表示运动停止。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

5.5.2 状态查询函数

int check_status（int ch）；/*检查一个轴的状态值*/
int check_done（int ch）；/*检测一个轴的运动是否完成*/
int check_limit（int ch）；/*检测一个轴指定的限位开关是否闭合*/
int check_home(int ch)；/*检查一个轴是否已经到达原点开关位置*/
int check_SD(int ch)；/*检查一个轴的外部减速信号*/
int check_alarm(int ch)；/*检查一个轴的外部报警信号*/
int get_cmd_counter()；/*用于获取当前正在执行的运动指令计数*/

函 数 名：check_status
目　　的：用 check_status 函数读取并返回一个轴的状态值。
语　　法：int check_status（int ch）；
　　　　　ch：所读取状态的轴号。
调用例子：ch_status=check_status（2）；
描　　述：函数 check_status 读取指定轴的状态。MPC08 控制卡每个轴都有 1 个 32 位（双字）的状态值，用于查询轴的工作状态。该双字中每位（bit）的含义如下图所示。
返 回 值：如果调用成功，check_status 返回指定轴的状态值，在出错时返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

D31～D27		
D26	0：OFF；1：ON	原点开关 ORG 的状态
D25	0：OFF；1：ON	正限位开关 EL+的状态
D24	0：OFF；1：ON	负限位开关 EL-的状态
D23～D16		
D15	0：OFF；1：ON	报警信号 ALM 的状态
D14		
D13	1：ALM 信号；0：非 ALM 信号	运动停止原因标志
D12		
D11		
D10	1：ORG 信号；0：非 ORG 信号	运动停止原因标志
D9	1：EL_P 信号；0：非 EL_P 信号	运动停止原因标志
D8	1：EL_N 信号；0：非 EL_N 信号	运动停止原因标志
D7	0：运动停止；1：正在运动	轴运动状态
D6		
D5D4	00：停止状态；01：匀速状态；10：加速状态；11：减速状态	轴运动状态
D3		

D2		
D1	0：ON；1：OFF	减速信号 SD 的状态
D0		

函 数 名：check_done
目　　的：用 check_done 函数来检查指定轴的运动是否已经完毕。
语　　法：int check_done（int ch）；
　　　　ch：所检查的轴号。
描　　述：函数 check_done 检查指定轴是在运动中还是在静止状态。
返 回 值：如果指定轴正在运动状态，check_done 返回 1，如果指定轴正在静止状态，check_done 返回 0，函数调用失败返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：check_limit
目　　的：用 check_limit 函数来检查一个轴是否已经到达限位开关位置。
语　　法：int check_limit（int ch）；
　　　　ch：所检查的轴号；
调用例子：status=check_limit（1）；
描　　述：函数 check_limit 用于检测指定轴的限位开关，并返回指定轴的位置是否已经到达限位开关位置及到达哪一个方向上的限位开关位置。
返 回 值：如果 check_limit 返回 1 表示到达正向限位开关位置，返回-1 表示到达负向限位开关位置，返回 0 则表示未到达限位开关位置，返回 2 表示同时到达正向限位和负向限位，若调用出错则返回-3。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP

函 数 名：check_home
目　　的：用 check_home 函数来检查一个轴是否已经到达原点开关位置。
语　　法：int check_home（int ch）；
　　　　ch：所检查的轴号；
调用例子：status=check_home（1）；
描　　述：函数 check_home 用于检测指定轴的原点开关，并返回指定轴的位置是否已经到达原点开关位置。
返 回 值：如果 check_home 返回 1 表示到达原点开关位置，返回 0 则表示未到达限位开关位置，若调用出错则返回-3。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP

函 数 名：check_SD
目　　的：用 check_SD 函数来检查指定轴的外部减速信号。
语　　法：int check_SD（int ch）；
　　　　ch：所检查的轴号。
描　　述：函数 check_SD 检查指定轴的外部减速信号。
返 回 值：如果指定轴的外部减速信号有效，check_SD 返回 1，若没有外部减速信号，则 check_SD 返回 0，若调用出错则返回-3。

系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：**check_alarm**
目　　的：用 check_alarm 函数来检查外部报警信号。
语　　法：int check_alarm（int ch）；
　　　　　ch：轴号。
描　　述：函数 check_alarm 检查外部报警信号。
返 回 值：如果外部报警信号有效，check_alarm 返回 1，若没有外部报警信号，则 check_alarm 返回 0，函数调用失败返回-3。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：**get_cmd_counter**
目　　的：用于获取当前正在执行的运动指令计数。
语　　法：int get_cmd_counter（）；
调用例子：cmdcounter=get_cmd_counter（）；/*读取当前正在执行的运动指令计数将其保存在变量 cmdcounter 中*/
描　　述：若要知道当前正在执行第几条运动指令，可通过该函数查询。运动指令计数从初始化完成后的 0 开始，随着执行的运动指令（即能产生运动的指令，不包括 set_conspeed 等设置指令）递增，可以调用 reset_cmd_counter() 函数进行清零。
返 回 值：如果调用成功，则返回值为当前正在执行的运动指令计数值，否则返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：reset_cmd_counter

5.6 I/O 口操作函数

在 MPC08 运动函数库中提供了 5 个 I/O 口操作函数：
int checkin_byte(int cardno)；/*从板卡 I/O 口以两个字节方式读入*/
int checkin_bit(int cardno,int bitno)；/*检查板卡 I/O 口某位状态*/
int outport_byte(int cardno,int bytedata)；/*向板卡 I/O 口 24 位同时输出*/
int outport_bit(int cardno,int bitno,int status)；/*从板卡 I/O 口以每一位方式输出*/
int check_SFR(int cardno)；/*读取外部减速、限位和原点信号状态*/

函 数 名：**checkin_byte**
目　　的：从板卡的 I/O 口以两个字节方式读入输入开关量状态。
语　　法：int checkin_byte(int cardno)；
　　　　　cardno：卡编号；
描　　述：MPC08 卡提供 16 个通用的光电隔离输入口，供用户使用。通过该函数可以读入这 16 个输入口的状态。接线见 MPC08 接口一节内容。若调用 enable_sd 函数将外部减速信号设置为无效，则通用输入口可增加到 20

个。

返 回 值：返回输入口的状态，返回值的 D0 到 D16 位对应 16 个输入口，该位为 1 表示输入口处于 ON 状态，为 0 表示该输入口处于 OFF 状态；如果出错则返回-1。

系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP

参　　见：checkin_bit，enable_sd

函 数 名：checkin_bit

目　　的：读入板卡的 I/O 口某一位开关量状态。

语　　法：int checkin_bit(int cardno,int bitno)；

cardno：卡编号；

bitno：表示第几位，取值范围为 1~16。

描　　述：MPC08 卡提供 16 个通用的光电隔离输入口，供用户使用。通过该函数可以读入某一个输入口的状态。接线见 MPC08 接口一节内容。若调用 enable_sd 函数将外部减速信号设置为无效，则通用输入口可增加到 20 个。

返 回 值：返回输入口的状态，返回值为 1 表示输入口处于 ON 状态，为 0 表示该输入口处于 OFF 状态；如果出错则返回-1。

系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP

参　　见：checkin_byte，enable_sd

函 数 名：outport_byte

目　　的：从板卡的 I/O 口以 16 位方式设置输出开关量状态。

语　　法：int outport_byte(int cardno,int bytedata)；

cardno：卡编号；

bytedata：状态字节，各位对应各输出口；

描　　述：MPC08 卡提供 16 个通用的光电隔离输出口，供用户使用。通过该函数可以设置这 16 个输出口的状态。接线见 MPC08 接口一节内容。

返 回 值：正确设置返回 0；如果出错则返回-1。

系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP

参　　见：outport_bit

函 数 名：outport_bit

目　　的：设置某个输出口开关量状态。

语　　法：int outport_bit(int cardno,int bitno,int status)；

cardno：卡编号，取值范围从 1 到卡最大编号；

bitno：表示第几个输出口，取值范围为 1~16。

Status：设置的状态；(1：ON；0：OFF)

描　　述：MPC08 卡提供 16 个通用的光电隔离输出口，供用户使用。通过该函数可以设置某一个输出口的状态。接线见 MPC08 接口一节内容。

返 回 值：正确设置返回 0；如果出错则返回-1。

系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP

参　　见：outport_byte

函 数 名：**check_SFR**

目　　的：用于读取外部减速、限位和原点信号状态。

语　　法：int check_SFR(int cardno)；

cardno：卡号；

调用例子：check_SFR（1）；/*读取 1 号卡的外部减速、限位及原点信号*/

描　　述：调用该函数读取卡的外部减速、限位、原点信号。如果将某轴的外部减速、限位、原点信号设置为无效，则对应的原点信号输入端口可作为通用输入口使用，这些端口的状态保存在一个 32 位寄存器中，每一位的定义如下图所示，使用函数 check_SFR 可读取相应端口状态。

返 回 值：返回外部开关量信号的状态，返回值的 D0~D16 位对应 17 个输入口，相应位为 1 表示输入口处于高电平状态，为 0 表示该输入口处于低电平状态；如果出错则返回-1。如果函数调用成功，则返回值为 0；否则若返回-1。

系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP

参　　见：enable_sd，enable_el，enable_org

D16	ALM	ALM 无效时作为通用输入口
D15	ORG4	ORG4 无效时作为通用输入口
D14	EL4+	EL4+无效时作为通用输入口
D13	EL4-	EL4-无效时作为通用输入口
D12	SD4	SD4 无效时作为通用输入口
D11	ORG3	ORG3 无效时作为通用输入口
D10	EL3+	EL3+无效时作为通用输入口
D9	EL3-	EL3-无效时作为通用输入口
D8	SD3	SD3 无效时作为通用输入口
D7	ORG2	ORG2 无效时作为通用输入口
D6	EL2+	EL2+无效时作为通用输入口
D5	EL2-	EL2-无效时作为通用输入口
D4	SD2	SD2 无效时作为通用输入口
D3	ORG1	ORG1 无效时作为通用输入口
D2	EL1+	EL1+无效时作为通用输入口
D1	EL1-	EL1-无效时作为通用输入口
D0	SD1	SD1 无效时作为通用输入口

5.7 其它函数

在 MPC08 运动函数库中提供了 10 个其它辅助功能函数：

int set_backlash（int ch，int backlash）/*设置由于机构换向形成间隙的补偿值*/

int start_backlash（int ch）；/*开始间隙补偿*/

int end_backlash（int ch）；/*终止间隙补偿（end backlash compensation）*/

int change_speed(int ch,double speed); /*运动中变速*/
int change_accel(int ch,double accel); /*运动中变加速度*/
int Outport（int portid,unsigned char byte）; /*对某个口地址输出一个字节*/
int Inport（int portid）; /*从某个口地址输入一个字节*/
int set_ramp_flag(int flag)；/*用于在多条运动指令连续执行时进行特殊升降速处理的设置*/
int get_lib_ver(long* major,long *minor1,long *minor2)；/*用于查询函数库的版本*/
int get_sys_ver(long* major,long *minor1,long *minor2)；/*用于查询驱动程序的版本*/
int get_card_ver(long cardno,long *type,long* major,long *minor1,long *minor2)；/*用于查询板卡的版本*/

函 数 名：set_backlash，start_backlash，end_backlash
目　　的：用 set_backlash 设置补偿由于机构换向形成间隙的补偿值。
用 start_backlash 开始补偿由于机构换向间隙而导致的位置误差。
用 end_backlash 停止补偿由于机构换向间隙而导致的位置误差。
语　　法：int set_backlash（int ch，int backlash）；
int start_backlash（int ch）；
int end_backlash（int ch）
ch：轴号；
backlash：由于机构换向形成的间隙值，单位为脉冲数；
调用例子：set_backlash（1，12）；
start_backlash（1）；
end_backlash（1）
描　　述：函数 set_backlash 设置一个补偿，以便消除由于机构换向形成的位置误差。函数 start_backlash（CH1）调用后，开始补偿。补偿值是通过调用 set_backlash 设置的。
返 回 值：如果设置成功，set_backlash、set_backlash 和 end_backlash 返回 0，否则返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
注　　释：set_backlash 函数仅是设置补偿值，真正的补偿值到调用函数 start_backlash 才起作用。函数 set_backlash 应在调用 start_backlash 前调用，否则系统采用缺省补偿值（缺省值为 20 个脉冲）。
参　　见：

函 数 名：change_speed
目　　的：用 change_speed 函数来实现运动中变速的功能。
语　　法：int change_speed（int ch,double speed）；
ch：轴号；
speed：变化到的速度。
描　　述：函数 change_speed 函数来实现运动中变速的功能，变速范围的最大值不能超过 set_maxspeed()所设定的值，变速范围的最小值必须大于 0。当以快速运动指令启动运动后，即可调用该函数在运动过程中实现变速。变

速时的加速度由运动指令函数调用前的 set_profile 函数参数决定。
返 回 值：调用正确返回 0，错误返回-1。
系 统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参 见：set_profile

函 数 名：**change_accel**
目 的：用 change_accel 函数来实现运动中变加速度的功能。
语 法：int change_accel（int ch,double accel）；
ch：轴号；
accel：变化到的加速度。
描 述：函数 change_accel 函数来实现运动中变加速度的功能，变速范围的最小值必须大于 0。当以快速运动指令启动运动后，即可调用该函数在运动过程中实现变加速度。
返 回 值：调用正确返回 0，错误返回-1。
系 统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参 见：set_profile

函 数 名：**Outport**
目 的：用 Outport 函数来对 PC 机口地址进行写操作。
语 法：int Outport（int portid,unsigned char byte）；
int portid：口地址
unsigned char byte：输出的一个字节数据
描 述：函数 Outport 将一个字节的数据写到 portid 对应的口地址，如计算机并口。该函数与 MPC08 卡操作无关，通过该函数可方便地对 PC 机口进行读写操作。
返 回 值：函数正确执行返回 0，否则返回-1。
系 统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参 见：

函 数 名：**Inport**
目 的：用 Inport 函数来对口地址进行读操作。
语 法：int Inport（int portid）；
int portid：口地址
描 述：函数 Inport 读地址 portid 对应的输入口数据，如计算机并口。该函数与 MPC08 卡操作无关，通过该函数可方便地对 PC 机口进行读写操作。
返 回 值：返回读取的内容。
系 统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参 见：

函 数 名：**set_ramp_flag**
目 的：用于在多条运动指令连续执行时进行特殊升降速处理的设置。
语 法：int set_ramp_flag (int flag);
int flag：下一条运动指令特殊升降速标志

描　　述：当一段运动轨迹由多个微线段构成时，要求微线段间停顿时间尽可能短，第一条微运动有升速过程（无降速过程），中间微运动无升降速，最后一条微运动无升速过程，但有降速过程。这时，在第一条运动指令前调用库函数 set_ramp_flag，函数参数为 1，如：set_ramp_flag(1)；在最后一条运动指令前调用库函数 set_ramp_flag，函数参数为 2，如：set_ramp_flag(2)即可。
调用例子：err= set_ramp_flag (1);　/*设置下一条指令为特殊处理的第一条指令*/
　　err= set_ramp_flag (2);　/*设置下一条指令为特殊处理的最后一条指令*/
返 回 值：如果设置成功，则返回值为 0，否则返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：6.2 节多指令连续运动时的升降速处理

函 数 名：get_lib_ver
目　　的：用于查询函数库的版本。
语　　法：int get_lib_ver(long* major,long *minor1,long *minor2)；
long * major：返回的主版本号
long * minor1：返回的次版本号 1
long * minor2：返回的次版本号 2
调用例子：get_lib_ver(&major, &minor1, &minor2)；
描　　述：该函数可以查询运动控制卡函数库的版本号，函数库的版本号必须与驱动程序的版本号相同。
返 回 值：0。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：get_sys_ver
目　　的：用于查询驱动程序的版本。
语　　法：int get_sys_ver(long* major,long *minor1,long *minor2)；
long * major：返回的主版本号
long * minor1：返回的次版本号 1
long * minor2：返回的次版本号 2
调用例子：get_sys_ver(&major, &minor1, &minor2)；
描　　述：该函数可以查询运动控制卡驱动程序的版本号。
返 回 值：成功调用返回 0，否则返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：get_card_ver
目　　的：用于查询板卡的版本。
语　　法：int get_card_ver(long cardno,long *type,long* major,long *minor1,long *minor2)；
long cardno：卡号
long * type：卡类型号

long * major：返回的主版本号
long * minor1：返回的次版本号 1
long * minor2：返回的次版本号 2
调用例子：get_card_ver(1，&type, &major, &minor1, &minor2)；
描　　述：该函数可以查询运动控制卡的类型和版本号。
返 回 值：成功调用返回 0，否则返回-1。
系　　统：WINDOWS98、WINDOWS 2000、WINDOWS XP
参　　见：

6 常见问题及解决方法

6.1 基本功能及实现方法

6.1.1 函数库初始化

在应用程序初始化部分添加如下代码，这些代码对每一个应用程序均是必须的：

```
int Rtn;
…
Rtn=auto_set();
If(Rtn<=0)
{
    //错误：自检错误
}
Rtn=init_boad();
If(Rtn<0)
{
    //错误：初始化错误
}
…
```

以上初始化完成后，各轴速度及模式设置缺省值如下：

参数类型	参数	缺省值	修改函数
常速运动参数	各轴常速度	2000	set_conspeed
	矢量常速度	2000	set_vector_conspeed
快速运动参数	低速	2000	set_profile
	高速	8000	
	加/减速度	80000	
	矢量低速	2000	set_vector_profile
	矢量高速	8000	
	矢量加/减速度	80000	
	升降速类型	梯形	
模式及状态	脉冲输出方式	脉冲/方向方式	set_outmode
	输出信号逻辑	负逻辑	set_outmode
	回原点运动时检测原点信号方式	仅检测原点接近开关信号	set_home_mode
	指令执行方式	立即方式	

之后可根据需要进行模式切换，这些函数需要根据硬件实际情况进行调用设置，缺省设置及函数各参数具体含义见函数描述一章。若使用缺省设置，则程序中可以不调用这些函数。

```
set_outmode(1,1,0);        //脉冲输出模式设置
```

```
set_home_mode(1,0);     //回原点时检测原点信号的方式
```

6.1.2 简单的定位运动

速度和加减速度单位如下：

pps=每秒脉冲数(pulse per second)；

ppss=每秒 pps（pps per second）；

1）以下一段代码使某根轴按常速运动一段距离：

```
set_conspeed(1,1000);        //设置 1 轴常速度为 1000
con_pmove(1,10000);          //使 1 轴按照 1000 的常速度运动 10000 个脉冲
```

2）以下一段代码使某根轴按梯形速度运动一段距离：

```
set_profile(1,0,1000,1000);  //设置 1 轴低速为 0，高速为 1000，加速度为 1000
fast_pmove(1,10000);         //使 1 轴按照设置的梯形速度运动 10000 个脉冲
```

3）三轴同时运动，各以不同的速度运动不同的距离

```
set_conspeed(1,1000);
set_conspeed(2,2000);
set_conspeed(3,3000);
con_pmove(1,10000,2,30000,3,20000);
```

请注意，如果调用 fast_pmove 函数，那么应先调用 set_profile 函数设置所需要的梯形速度，否则，电机运转将使用缺省参数：set_profile（axis, 2000, 8000,80000）。对于 con_pmove 运动，应先调用 set_conspeed 函数设置常速，否则 conspeed 的缺省值是 2000pps。

6.1.3 简单的连续运动和回原点运动

连续运动函数使电机按照一个特定的速度一直运转，直到调用 sudden_stop 或 decel_stop 使其停止，或者遇到限位信号、外部报警信号等。这些函数被称之为 vmove 函数，是因为电机以一特定的速度(velocity)运动。下面一段代码为连续运动的例子程序：

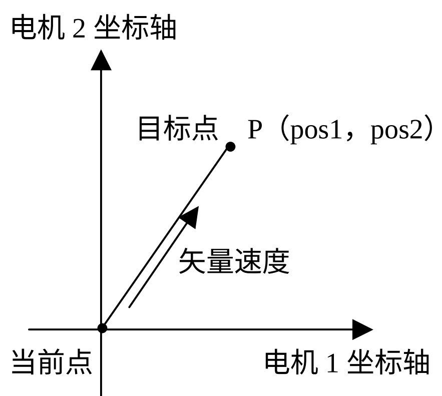
```
set_conspeed (1,1000);  //设置 1 轴常速度
con_vmove(1,1);         //令 1 轴以常速连续运转
```

调用回原点运动函数可以使机床或运动台返回原点，这些函数以常速或梯形速度运动，直到 MPC08 卡接收到相应的原点接近开关发出的信号为止。下面是一段调用 con_hmove 的代码，达到原点时运动将自动停下来。

```
set_conspeed (1, 1000); //设置 1 轴常速度
con_hmove(1,1)；        //令 1 轴以常速回原点
```

6.1.4 多轴插补运动

多轴插补运动只有线性运动，它们的运动速度由矢量速度（常矢量速度或梯形矢量速度）决定，各轴的速度为矢量速度在各轴上的分量。



1）下面一段代码使两轴以常速度作直线插补

```
set_vector_conspeed(1000);   //设置矢量常速度
con_line2(1,5000,2,2000);    //轴 1 移动 5000 个脉冲，轴 2 移动 2000 个脉冲
```

2）下面一段代码使两轴以梯形速度作直线插补

```
set_vector_profile(600,3000,10000);//设置矢量梯形速度
fast_line2(1,5000,2,10000);   //轴 1 移动 5000 个脉冲，轴 2 移动 10000 个脉冲
```

注意：直线运动可以分为两类：常速模式（con_line）和快速模式(fast_line)。上述代码演示的就是一个以常速和梯形速度走直线插补运动的例子。在这种模式下，矢量低速、矢量高速和矢量加速度应在调用前设定，否则这些参数将取缺省值。在常速模式下，只需设置常矢量速度。

6.2 多指令连续运动时的升降速处理

6.2.1 功能说明

在多条运动指令连续执行时，可进行特殊升降速处理，即第一条运动指令只有升速过程而无降速过程，中间运动指令既无升速过程也无降速过程，最后一条指令只有降速过程而无升速过程。

6.2.2 实现方法及应注意的问题

1）实现方法

对于要进行特殊升降速处理的一批指令，在第一条运动指令前调用库函数 set_ramp_flag，函数参数为 1，如：set_ramp_flag(1)；在最后一条运动指令前调用库函数 set_ramp_flag，函数参数为 2，如：set_ramp_flag(2)即可。

示例代码如下（注意，这段代码只是说明 set_ramp_flag 的使用，运动指令连续写出，实际应用时应先判断上一运动指令结束才能发下一条运动指令）：

```
int i;
……
set_ramp_flag(1);                    //设置特殊升降速处理起始标记
fast_line2(1,20000,2,20000);
con_line2(1,20000,2,20000);
con_line2(1,-20000,2,-20000);
set_ramp_flag(2);                    //设置特殊升降速处理结束标记
con_line2(1,-20000,2,-20000);
……
```

2）注意的问题

（1） set_ramp_flag(1)后第一条运动指令必须是快速运动，其后的所有运动指令均为常速运动指令，包括调用 set_ramp_flag(2)后发出的最后一条运动指令。

（2） 请严格按照使用方法中代码顺序及位置调用 set_ramp_flag()库函数，并传递正确的参数，否则将可能出现不正确的升降速。

（3） set_ramp_flag(1)和 set_ramp_flag(2)必须成对使用，否则可能会产生意外情况。

6.3 运动变速

用 change_speed 函数可以很容易实现运动中变速，以下代码演示实现运动中变速的问题。

定义全局变量

```
double CurSpeed=0;
double MaxSpeed=100000;
```

在“启动”按钮的响应函数中增加如下代码：

```
set_maxspeed(1,MaxSpeed);
set_vector_conspeed(100,1000,1000);
fast_pmove(1,1000000);
```

在“升速”按钮的响应函数中增加如下代码：

```
CurSpeed=CurSpeed+1000;
If(CurSpeed>MaxSpeed) CurSpeed=MaxSpeed;   //限定变速范围不超过最大值
change_speed(1,CurSpeed);
```

在“降速”按钮的响应函数中增加如下代码：

```
CurSpeed=CurSpeed-1000;
change_speed(1,CurSpeed);
```

先单击“启动”按钮启动运动，之后每单击一次“升速”按钮，当前速度增加1000；每单击一次“降速”按钮，当前速度减小1000。

6.4 正确判断前一个运动指令是否执行完毕

check_done()函数可以在立即方式下运动指令后直接判断运动是否停止。示例代码如下：

```
……
con_pmove(1,1000);             //第一条运动指令：1轴发出1000个脉冲
while ( check_done(1) = = 1) ;//循环判断1轴是否运动完毕，运动结束后执行下
                                一条运动指令
con_pmove ( 1,2000 ) ;         //第二条运动指令：1轴发出2000个脉冲
……
```

6.5 MPC08卡安装过程中常见问题及解决

6.5.1 Windows起动后未出现检测到PCI Card的信息

请仔细先后按以下几个方面检查：

(1) 检查卡与插槽是否接触良好，可更换其他PCI插槽再试一下；

(2) 若计算机上曾使用过MPC08运动控制卡（安装过驱动程序），且未重装过操作系统，并且当前使用的PCI插槽以前曾插入并正确使用过MPC08运动控制卡，则可能不会出现提示；若当前使用的PCI插槽以前从未插入并正确使用过MPC08运动控制卡，则会出现提示，但系统会自动安装驱动程序，不需要安装光盘。按提示重新起动计算机后即可完成安装。

(3) 查看系统设备管理器，若安装过MPC08运动控制卡的驱动程序，则应该出现“（StepServo）运动控制卡”一栏，展开该栏后应出现“MPC08 Driver”一项，表示MPC08运动控制卡设备。若该项图标上出现一个小“感叹号”，则表示该设备驱动程序未正确安装，此时可以从设备管理器中删除该项，并且删除Windows驱动程序目录（若Windows安装在C:\Windows目录下，则该目录为C:\Windows\System32\drivers）下的MPC08.sys，然后重新起动计算机，让计算机重新检测并安装驱动程序。

（4） 取下其它板卡，如声卡、网卡等，只保留显卡和 MPC08 运动控制卡后再起动计算机试一下，以避免因与其它卡产生冲突导致无法正确识别。

6.5.2 出现了检测到 PCI Card 的信息，但无法正确加载驱动程序

插入 MPC08 卡并起动计算机后，系统提示检测到 PCI Card 的信息，并起动“ 添加新硬件向导 ”对话框，但在 CDROM 上搜索不到驱动程序，出现此种情况，请按照如下步骤进行检查：

（1） 计算机上许多基于 PCI 总线的设备，如 PCI 声卡、网卡（包括主板集成声卡、网卡、Modem）等，在第一次将新设备插入系统后，都会出现该提示，不同的设备需要不同的驱动程序进行安装，可能系统此时找到的 PCI Card 并非 MPC08 运动控制卡而是其他设备，因此用 MPC08 安装光盘将会出现找不到驱动程序的提示。解决办法是：先关闭计算机并取下 MPC08 卡，起动计算机后按提示先将其他设备的驱动程序安装完成，特别是集成主板，上面集成的设备比较多，应用主板驱动光盘依次进行安装，直到每次起动计算机后不再出现找到 PCI Card 的信息为止。然后关闭计算机，插入 MPC08 卡后起动计算机，按提示完成 MPC08 的驱动程序安装。

（2） 若按照前一步骤处理后仍出现问题，请检查 MPC08 卡是否插好，特别是金手指部分是否有氧化现象或比较脏，可用无水酒精进行擦拭，待干后再插入计算机。因为此种情况会导致系统无法正确读取卡上的配置信息，也就无法正确匹配并加载驱动程序。

（3） 若前两个步骤仍然无法解决该问题，则可能是与其他设备冲突，此时请去掉其它卡或换一台计算机再试。

6.5.3 驱动程序安装正确，但无法正常发脉冲

请仔细按以下几个方面检查：

（1） 检查卡是否正确插入计算机 PCI 插槽；

（2） 查看系统设备管理器，以确信驱动程序已经正确安装。即设备管理器的设备列表是否出现“ （StepServo）运动控制卡 ”一栏，展开该栏应出现“ MPC08 Driver ”一项，并且图标上不应有小“ 感叹号 ”出现。

（3） 用安装光盘上提供的 Demo 安装后进行测试，并注意观察发脉冲期间卡上的指示灯是否变亮，若变亮，则检查转接板与卡的连接线是否插好，以及转接板连线是否正确，可测量转接板上的输出信号。

（4） 若指示灯不亮，则断电后去掉卡与转接板的连线后运行 Demo；

（5） 重新起动计算机后直接用 Demo 进行测试，因为若用户在调试自己的程序时因意外导致程序非正常退出（如非法操作或有某些线程没有正常终止而退出了程序主界面），则再次运行程序时可能导致卡无法正常工作；

（6） 若问题仍然存在，请去掉计算机中其他板卡或换一台其他配置的计算机再试，若这样能正常工作，则可能是与其它设备产生了冲突。特别是原装品牌机，因多采用集成主板，且主板集成设备驱动程序不规范，更容易产生冲突，因此从稳定性考虑，应优先选用知名厂家生产的非集成主板。

（7） 调用错误代码获取函数，根据返回的错误代码进行分析；

6.6 其它问题及解决方法

6.6.1 运行 EXE 文件时系统显示找不到 DLL 文件

可能是用户尚未正确安装 MPC08 软件，请按照“控制卡的安装”一章节内容安装软件。

注意：安装程序将把设备驱动程序 MPC08.sys 安装到 Windows 系统 System32 目录下（若 Windows 安装在 C 盘 Windows 目录下，则该目录为 C:\windows\system32\Drivers），函数库（动态链接库 DLL）则被安装到 Windows 系统目录下（C:\Windows\system），其余文件安装到在安装过程中指定的安装目录下。这样便可以在任何目录下使用函数库。但在升级函数库时应注意。

6.6.2 如何将开发的软件系统制作成安装程序后发行给最终用户

在制作安装程序时，请将如下文件打包进安装程序：

（8） Mpc 动态链接库文件 MPC08.dll

（9） VC 动态链接库文件 msvcrt.dll 和 mfc60.dll

同时，还应提供设备驱动程序 MPC08.sys 及安装程序 MPC08.inf 给用户。

6.6.3 软件能够正常启动，但无法产生运动

请仔细按以下几个方面检查：

（1） 电机、驱动器等执行机构是否完好；

（2） 执行机构与计算机是否已经正确连接；

（3） 板卡是否已经插入计算机；

（4） 软件是否调用了初始化函数进行了初始化，通过初始化函数返回值判断初始化是否成功；

（5） 软件中运动指令参数是否正确；

（6） 调用错误代码获取函数，根据返回的错误代码进行分析；

6.6.4 如何升级函数库

请您经常访问本公司的网站（http://www.step-servo.com）以下载获取最新版本的驱动程序及函数库，新版本函数库将会保持与旧版函数库已有函数的兼容，并根据需要增加新的函数。

若您获得一套最新的函数库，您可以按照以下方法对您的旧函数库进行升级：

（1） 检查您获得的函数库是否完整。完整的函数库至少应包含 MPC08.sys 和 MPC08.dll 两个文件；

（2） 关闭与 MPC08 相关的正在运行的所有程序；

（3） 将这两个文件拷贝覆盖到相应的 Windows 系统目录下，如：假设 Windows 安装在 C:盘 Windows 目录下，则将 MPC08.sys 拷贝覆盖到 C:\windows\system32\drivers 目录下；将 MPC08.dll 拷贝覆盖到 C:\windows\system 目录下。

若新版函数新增了库函数，您要能使用新增函数，还应当更新工程中的库函数声明文件（如 C 中的头文件，VB 中的库函数声明模块文件）。

6.6.5 减速、原点信号的使用

在某个轴的梯形速度（由 set_profile 设置）运动过程中，如碰到减速开关，则该轴的运动速度将自动从高速减到低速，并保持低速运行。一般情况下，减速开关与原点接近开关配合使用，以提高在高速回原点时的定位精度。将减速开关安装在原点接近开关的前面，在高速回原点时，运动机构先碰到减速开关，使之减速，并以低速靠近原点接近开关，在这个过程中，减速开关信号应保持闭合有效，在到达原点位置时，停止运动。如果没有减速开关，在高速回原点时，碰到原点接近开关将立即停车，由于运动机构的惯性、原点接近开关的有效工作范围等因素，将会降低回原点的定位精度，并可能给机械造成冲击。

注意：这里的正、负向是指控制卡发送脉冲的方向，可能与运动机构的实际运动方向并不一致。

6.6.6 如何提高速度精度

在 MPC08 卡的使用中，有时发现在运动时用 get_rate 读取的频率与设置的脉冲频率差别较大，其原因如下：

MPC08 卡的输出脉冲频率由两个变量控制：脉冲分辨率和倍率，两者的乘积即输出的脉冲频率。由于倍率寄存器长度是有限的，即 13 位（最大值为 8191），如果要达到 2400KHz 的最大输出脉冲频率，脉冲分辨率应为（2400000/8191）=293Hz，如果实际使用的脉冲频率为 100Hz，显然 MPC08 卡只能输出一个分辨率的脉冲频率（即 293Hz）。为了解决这个问题，可以调用 set_maxspeed 设置需要达到的最大输出脉冲频率。比如：set_maxspeed（1，1000），设置后脉冲分辨率将被重新设置，为（1000/8191）=0.12Hz，这样就能满足低速时速度精度的问题。注意：MPC08 卡的最高分辨率可以达到 0.01Hz，但此时 MPC08 卡的最大输出频率只能达到 81.91Hz。

6.6.7 如何实现方向信号超前于脉冲信号

某种品牌的步进电机驱动器在控制时序上要求方向信号要比脉冲信号超前 500 微秒，用 MPC08 卡控制时出现这种现象：当发出反转指令时，电机会向原来的方向转动一点，然后才反转，造成位置不准。为什么会出现这种现象，怎么处理？

有些步进电机的驱动器在控制时序上要求方向信号要比脉冲信号超前一定时间（几十到几百微秒），否则将工作不正常。而 MPC08 卡的脉冲信号和方向信号基本上是同时发出的，所以当发出反转指令时，驱动器的方向信号还没完全翻转时就接收到了脉冲信号，故而电机会向原来的方向转动一点，在驱动器的方向信号完全翻转后，电机才反转。对于这种驱动器，在调用运动指令前，先设置将要运转的方向：set_dir（ch, dir），延时足够的时间，确保驱动器的方向信号稳定后，再调用运动指令。注意：在运动过程中不要调用 set_dir，否则会导致电机突然反转。

6.6.8 多卡共用问题

每块 MPC08 卡最多可以控制四轴的运动，我们的控制系统中需要控制五个轴，能否使用 MPC08 卡来完成？应该怎样来用？

尽管每块 MPC08 卡最多控制四个轴，但 MPC08 卡支持多卡共用，也就是说，在 PC 机上可以插入几块卡，而在编程时，把它们当成一个整体对待，各卡之间的轴与同一块卡上的轴也同样能够进行联动或插补，这样就能实现多于四轴的运动控制。对于五轴控制，可以用两块 MPC08 卡。一台计算机上可同时使用的 MPC08 卡的数目取决于 PCI 扩展槽的数目。

6.7 如何避免与其他设备的冲突

MPC08 运动控制卡基于 PCI 总线，配合 Windows 操作系统支持即插即用，所有资源（I/O 地址）由系统自动配置，因而使用非常方便，且一般不容易出现资源冲突。但在一些极个别的特殊情况下，也可能出现设备资源冲突，导致控制卡无法正常工作，如出现驱动程序无法正常加栽，运动指令出现比较明显的延迟现象或根本无法发出等，这种情况一般是由于 IO 空间分配失败导致，

为避免潜在的资源冲突以及稳定性需要，在配置 PC 机时尽量不要选用集成设备比较多的集成主板。

7 函数索引

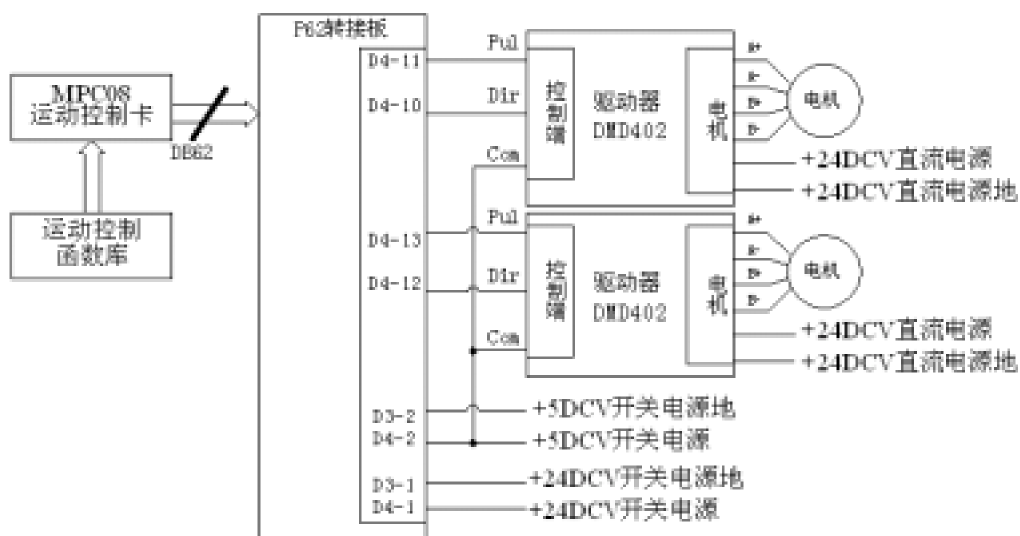
8 附录

8.1 两轴步进控制系统示例

8.1.1 系统配置

1. 混合式步进电机：成都步进公司 DM4240A（1.8°，0.32N.m）；
2. 驱动器：成都步进公司 DMD402（最大细分 128，峰值电流 2A）；
3. 上位控制：MPC08；
4. 直流开关电源：24DCV（10A），5DCV（1A）。

8.1.2 控制电路接线图



*关于步进电机 DM4240A 和驱动器 DMD402 的应用请参考本公司相应的使用说明书。

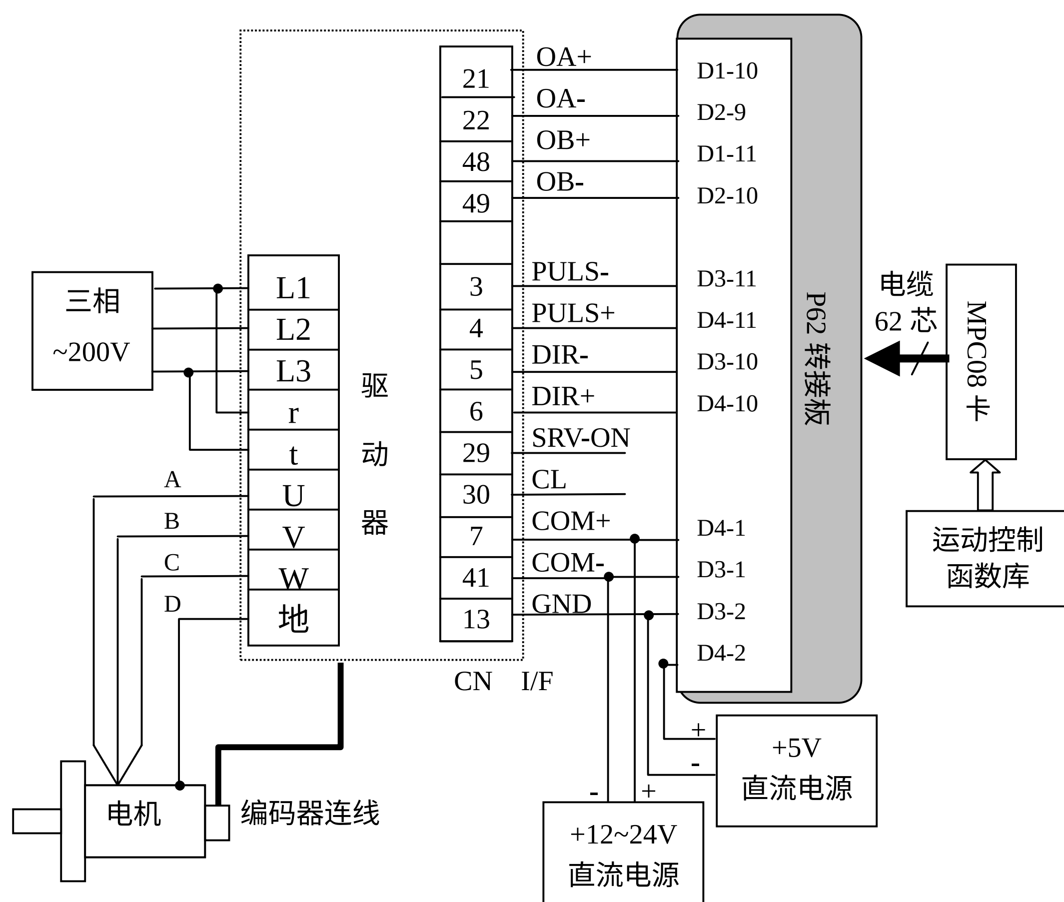
8.2 单轴数字式伺服控制系统示例

全数字式交流伺服系统（例如：富士 FALDIC-β系列或松下 MINAS A 系列）可以接收脉冲和方向信号来控制其运动过程，因此，可以用 MPC08 卡来实现对伺服电机系统的控制。下面以单轴伺服控制系统为例简要说明 MPC08 与伺服系统之间的连接方法。

8.2.1 系统配置

1．伺服系统：松下 MINAS A4 系列全数字式交流伺服系统任意型号；
2．控制卡：MPC08；
3．直流开关电源：24DCV（1A），5DCV（1A）。

8.2.2 控制电路接线图



MPC08 与 MINAS A4 系列的接线图

*有关松下全数字交流伺服系统的应用请参考其说明书。

8.3 PC 打印机口用作 I/O 口

PC 打印机接口为并行 I/O 口，与其它通用 I/O 口性质完全一样。它由一个 25 芯的 D 型接口提供 TTL 输入和输出信号，其引脚定义如下：

引脚	I/O 方向	打印机中的功能	信号说明
1	Out	-STROBE	数据输出触发
2~9	Out	Data bits（D0~D7）	输出数据到打印机
10	In	-ACK	应答
11	In	BUSY	忙
12	In	PE	纸尽
13	In	SLCT	打印机选择
14	Out	-Auto FD XT	自动回车换行
15	In	-ERROR	出错
16	Out	-INIT	初始化打印机
17	Out	-SLCTIN	选择数据输出到打印机
18~25	-	Commond ground	公共地

在上表中的输入输出信号共占用 PC 机的三个 I/O 地址，对于 LPT1（并行打印机口 1）而言，这三个 I/O 地址的各位（bit）定义如下：

I/O 地址	**信号种类**	**位**	**功　　能**	**DB25 引脚**
378H（输出）	数据信号	D0~D7	输出至打印机	Pin2~Pin9
379H（输入））	状态信号	D0~D2 D3 D4 D5 D6 D7	没使用 -ERROR SLCT PE -ACK BUSY	 Pin15 Pin13 Pin12 Pin10 Pin11
37AH（输出）	控制信号	D0 D1 D2 D3 D4 D5~D7	-STROBE -Auto FD XT -INIT -SLCTIN 中断允许（IRQ7） 没使用	Pin1 Pin14 Pin16 Pin17

由上表可见，一个打印机口，总共有 12 根输出和 5 根输入可供使用，一般能够满足需要少量 I/O 信号的场合。注意：379H 口的第 7 位（BUSY）在打印机接口电路中是从连接器经反相后接到总线上的；同样，37AH 口的 0、1、3 位也是经反相后接入总线的，在使用中应注意区分。

8.4 PC 机 I/O 地址分配

PC 机系统支持的端口地址范围是从 0~3FF，共 1024 个端口地址，有效译码地址信号是 A9~A0。其中前 512 个端口（0~1FFH）被系统板所占用，因此其它扩展板卡一般不应该使用这些端口；高端的 512 个端口（200~3FF）是为了扩展板卡预留的，但有一些已被标准扩展板所占用，例如：单显适配器占用 3B0H~3BFH，彩色图形适配器占用 3D0H~3DFH，因为这些扩展板卡是 PC 机的基本配置，所以这些端口用户不能使用。下表是 PC 机 I/O 端口地址的分配表，在表上，地址 200H 以后注明保留的或未出现在该表中的端口地址可以由用户使用。

PC 机 I/O 端口分配表

分类	地址范围（16 进制）	I/O 设备端口
系统板	000~01F	DMA 控制器 1
	020~03F	中断控制器
	040~05F	定时器/计数器
	060~06F	键盘
	070~07F	实时时钟
	080~09F	DMA 页面寄存器
	0A0~0BF	中断控制器 2
	0C0~0DF	DMA 控制器
	0F0~0FF	协处理器
	100~1EF	未用
	1F0~1F8	硬盘
I/O 通道	200~20F	游戏接口
	258~25F	Intel Above Board
	278~27F	并行打印口 2
	280~2DF	Ultimate EGA
	2E1	GPIB
	2E2~2E3	Data Acquisition
	2F8~2FF	串口 2
	300~31F	试验板
	360~36F	PC Network
	378~37F	并行打印口 1
	380~38F	SDLC 通信
	390~393	Cluster
	3A0~3AF	Binary Synchronous Communication
	3B0~3BF	单显适配器
	3C0~3CF	EGA
	3D0~3DF	彩色图形显示适配器
	3F0~3F7	软盘驱动器
	3F8~3FF	串口 1

8.5 PC 机中断线分配

每台 PC 机总共有 15 根有效的中断线，其中许多已被占用，下表列出了 PC 机中断线的一般分配情况，在使用时注意选用。一般来说，在工控系统中，有些外围设备并不需要，如网卡、打印机等，所以中断线 IRQ2、3、5、7、10、11、12、15 都能由用户使用，但为了保险起见，在使用某个中断线之前，最好核实它是否已被占用，以免发生冲突。

PC 机中断线分配表

IRQ 号	**用　　途**
IRQ2	等于 IRQ9，通常可用
	EGA Display Adapter
	PC Network
IRQ3	COM2
	PC Network
	Binary Synchronous Communication
	Cluster
	通常可用
IRQ4	COM1
	Binary Synchronous Communication
	SDLC
IRQ5	通常可用
IRQ6	Floppy Disk
IRQ7	打印口 LPT1
	Cluster
	通常可用
IRQ8	DOS 定时器
IRQ10	Open
IRQ11	Open
IRQ12	Open
IRQ14	Fixed Disk
IRQ15	Open